

FOR INFORMATION PURPOSE

DRAFT

**APPLICATION SECURITY ASSESSMENT
METHODOLOGY**



Version 1.0

November 22, 2002

**Center for Information Assurance Applications
Applications and Computing Security Division
5275 Leesburg Pike
Falls Church, VA 22041**

**(This document is for review. Comments, if any, can be sent to
JainD@ncr.disa.mil or KoehlerS@ncr.disa.mil)**

FOR INFORMATION PURPOSE

TABLE OF CONTENTS

	Page Number
1. INTRODUCTION	1
1.1 OBJECTIVE.....	1
1.2 BACKGROUND.....	1
1.3 SCOPE.....	2
1.3.1 Intended Audience	2
1.3.2 Assessment Environment	3
1.3.3 Assessment Methodology Updates.....	3
1.4 DOCUMENT STRUCTURE.....	3
2. TOOLKIT OVERVIEW.....	5
2.1 TOOL CATEGORIES	5
2.1.1 Web Application Assessment Tools.....	6
2.1.2 Database Assessment Tools	6
2.1.3 General-Purpose Assessment Tools.....	6
2.1.4 Developer Assessment Tools.....	6
2.2 TOOL OVERVIEW	7
2.2.1 SPIDynamics WebInspect (Web Application Assessment Tool).....	8
2.2.2 ISS Database Scanner (Database Assessment Tool)	8
2.2.3 eEye Digital Security Retina (General-Purpose Assessment Tool)	9
2.2.4 Flawfinder (Developer Assessment Tool).....	10
2.2.5 RATS (Developer Assessment Tool)	11
2.2.6 Splint (Developer Assessment Tool)	11
2.3 RUNNING TOOLS FROM TOOLKIT	12
3. ASSESSMENT PROCESS OVERVIEW.....	13
4. PRE-ASSESSMENT PHASE.....	15
4.1 APPLICATION CATEGORIZATION.....	16
4.2 VULNERABILITY AND REQUIREMENTS IDENTIFICATION.....	17
4.3 TOOLKIT CAPABILITY TO REQUIREMENTS MAPPING.....	17
4.4 TOOL SELECTION.....	18
4.5 SUPPLEMENTAL PROCEDURES AND ASSESSMENT PROCESS.....	18
4.6 TEST ENVIRONMENT PREPARATION	21

5. ASSESSMENT PHASE.....	23
5.1 SPIDYNAMICS WEBINSPECT	23
5.1.1 Tool Configuration	23
5.1.2 Tool Use.....	28
5.1.3 Interpreting Tool Output	34
5.2 ISS DATABASE SCANNER	36
5.2.1 Tool Configuration	36
5.2.2 Tool Use.....	38
5.2.3 Interpreting Tool Output	41
5.3 EEYE RETINA.....	42
5.3.1 Tool Configuration	42
5.3.2 Tool Use.....	48
5.3.3 Interpreting Tool Output	50
5.4 SPLINT	51
5.4.1 Tool Configuration	52
5.4.2 Tool Use.....	52
5.4.3 Interpreting Tool Output	53
5.5 FLAWFINDER.....	55
5.5.1 Tool Configuration	55
5.5.2 Tool Use.....	56
5.5.3 Interpreting Tool Output	57
5.6 RATS	59
5.6.1 Tool Configuration	59
5.6.2 Tool Use.....	60
5.6.3 Interpreting Tool Output	60
6. POST-ASSESSMENT PHASE	65
6.1 TAKE CORRECTIVE ACTIONS AND APPLY SUGGESTED FIXES	65
6.2 VALIDATE FUNCTIONALITY AND RERUN TOOL.....	66
6.3 CAPTURE LESSONS LEARNED	67
6.4 ESTABLISH A TESTING AND EVALUATION REGIME	67
7. CONCLUSION	68
APPENDIX A: ACRONYMS	A-1
APPENDIX B: REFERENCES	B-1
APPENDIX C: REQUIREMENTS - TOOLKIT CAPABILITY MAP	C-1

1. INTRODUCTION

1.1 OBJECTIVE

This document establishes an assessment process for application developers and security engineers to use as they design and assess security mechanisms in their applications. The methodology will assist application developers in identifying vulnerabilities in their applications and in validating application security requirements as defined in the *Recommended Standard Application Security Requirements* document, drafted as a precursor to this methodology. The assessment methodology will serve as a first step in ensuring that security is designed into existing applications and will aid application developers in identifying security flaws early in the life cycle of an application. The methodology can be used to assess and validate requirements in applications of various types, including stand-alone Web applications and Web applications that interoperate with “backend” databases and other legacy servers. The methodology presents ways to identify general vulnerabilities and common programming/coding errors for a variety of applications.

It is anticipated that the methodology presented herein will be used by not only developers to design security into applications during the development phase, but also certification and accreditation personnel in the future to validate an application’s security integrity. Furthermore, the methodology can be used to identify vulnerabilities and improve security currently designed into existing applications.

This methodology provides an overview of tools that can be used during the assessment process and explains how the tools have been categorized to best assess requirements against a specific application. The methodology addresses proper tool selection, configuration, and scanning techniques. Guidelines for interpreting assessment results are presented and ways to identify corrective actions are suggested.

1.2 BACKGROUND

The Defense Information Systems Agency (DISA) Application and Computing Security Division’s (API2) mission is to provide for the identification, development, system engineering, prototyping, provisioning, and implementation of various technologies supporting the defense-in-depth (DID) concept for multilayered protection of the global applications and computing infrastructure of the global information grid (GIG). API2 has defined a set of application security requirements and common vulnerabilities applicable to a majority of applications. These security requirements were documented in the *Application Security Developer’s Guide*. API2 identified a set of tools that can be used to identify application vulnerabilities and are grouped in the Application Security Assessment Toolkit.

This document presents a methodology to test for application security requirements and vulnerabilities presented in the *Recommended Standard Application Security Requirements* document. The assessment methodology uses assessment tools identified in the *Application Security Assessment Tools Market Survey* and suggests corrective actions where applicable based on guidance presented in the *Application Security Developer’s Guide*. This assessment methodology will help application

developers and test engineers establish an assessment process that can be used throughout the application's development life cycle.

1.3 SCOPE

The methodology presented herein defines a process that developers can use to test and validate applications of all types regardless of platform or operating system. The methodology can be used to assess server applications, client applications, and standalone applications where the application types range from Web, database and collaboration applications, and ultimately to legacy applications. At this point, the methodology was developed with a multitool approach in mind because no single tool exists to test all types of applications as well as validate all security requirements. Therefore, multiple tools must be used (thus, the development of the Application Security Assessment Toolkit) along with supplemental procedures when assessing the security of an application. The methodology predominately focuses on Web applications because of the maturity of tools available. This methodology may be applied to legacy applications once assessment tools are developed or modified to assess the security of legacy applications.

Note, however, that completion of this methodology does not guarantee perfect security in an application. The tools within the Toolkit, although quite comprehensive, will not validate all security requirements. Moreover, use of the Toolkit will not guarantee that all security vulnerabilities will be discovered within an application. In fact, all items within the Toolkit are continually being improved and updated by their respective manufacturers and vendors as new vulnerabilities are discovered and existing detection techniques are updated. As such, it is suggested that applications be assessed repeatedly for vulnerabilities throughout their life cycle.

1.3.1 Intended Audience

Application developers should use this methodology to learn how to identify and improve on unintentional vulnerabilities introduced during application development, thereby making future applications more secure.

Application developers and security engineers will use this methodology as a starting point for testing and identifying potential vulnerabilities in their applications and assessing the validity of security requirements and features designed into applications.

The intended audience for this methodology document shall be application developers, security engineers, certifiers and accreditors, and any other personnel involved in the application development and security testing process. It is assumed that the audience following the methodology presented in this document is already well versed in software development processes and testing practices associated with vulnerability assessments. The individuals implementing this methodology should have an understanding of, and general experience with, the target application(s). It is also assumed that they are familiar with the assessment tools contained in the Application Security Assessment Toolkit. For this document, this audience shall be referred to, in an all-inclusive sense, as "developers."

1.3.2 Assessment Environment

Application assessments using this methodology should be carried out in a test environment. This methodology recommends using assessment tools that are capable of stressing applications to their maximum design limits. Assessment tests performed can fill logs, databases, directories, and other data collectors with great quantities of test data. The testing performed can also introduce many errors into an application or Web site. The ability to restart applications or revert back to pretest baseline conditions is essential. It is strongly suggested that a comprehensive assessment be conducted only against development sites and applications, not against production environments.

This methodology can be used to assess production applications; however, it may not be possible to perform all tests. Careful configuration of the vulnerability assessment tools is required to ensure that the applications remain operational in a production environment.

1.3.3 Assessment Methodology Updates

This “living document” will be updated periodically as new assessment tools are developed and made available. The assessment methodology presented in the following sections is geared toward assessing applications using the scanning tools contained in the Application Security Assessment Toolkit, dated October 2002.

Although the assessment methodology is not all-inclusive, it will be as comprehensive as possible using the technologies now available. The general application assessment approach presented in the sections below should remain constant regardless of additions and changes to the Toolkit. If anything, the assessment methodology will be enhanced to address new security requirements, new capabilities, and new methods to check for additional security requirements as functionality is added to future versions of the Toolkit.

1.4 DOCUMENT STRUCTURE

This document consists of seven sections and three appendices. An overview of the sections is provided below.

- Section 1, Introduction, describes the objectives, scope, and structure of this document.
- Section 2, Toolkit Overview, describes the overall Toolkit concept and provides an overview of each tool.
- Section 3, Assessment Process Overview, provides a high-level discussion of the overall assessment methodology.
- Section 4, Pre-Assessment Phase, describes the necessary steps in the pre-assessment process.
- Section 5, Assessment Phase, discusses the procedures required to conduct an assessment using each tool from the current toolkit.
- Section 6, Post-Assessment Phase, describes the necessary steps in the post-assessment process.

- Section 7, Conclusion, summarizes and concludes this entire document.
- Appendix A, Acronyms, lists the acronyms used throughout this document.
- Appendix B, References, lists the references used throughout this document.
- Appendix C, Requirements-Toolkit Capability Map, facilitates mapping the current Toolkit's capabilities to the security requirements established in the *Recommended Standard Application Security Requirements* document.

2. TOOLKIT OVERVIEW

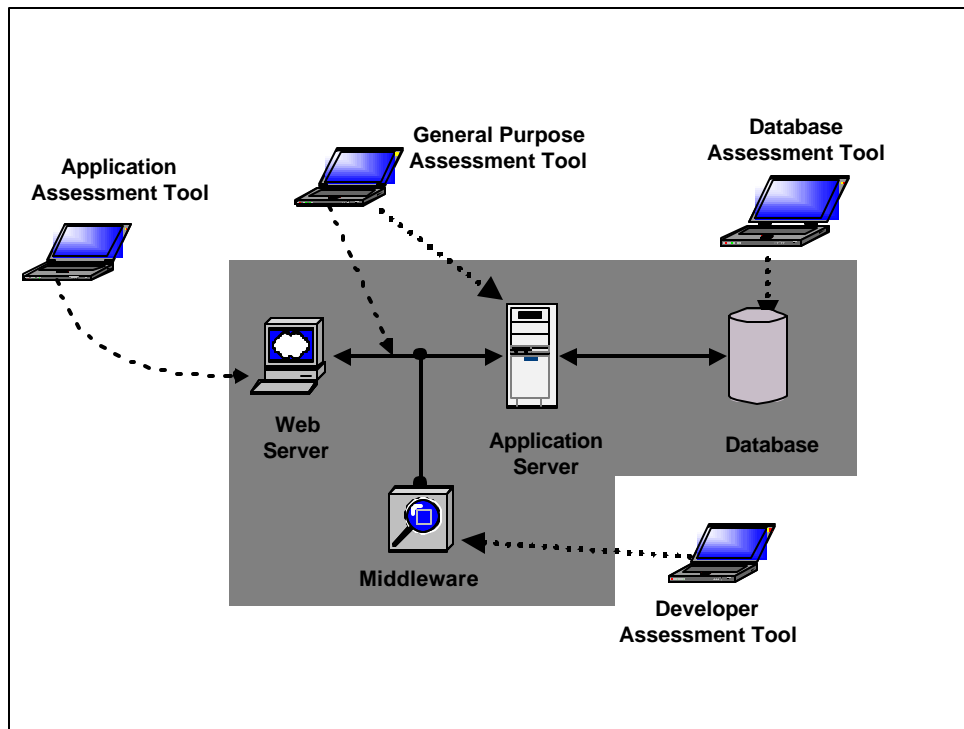


Figure 2-1. Application Security Assessment Toolkit Concept

2.1 TOOL CATEGORIES

The application developer is challenged to work in a space that encompasses many different platforms, hosts, and applications. This workspace, as illustrated by the gray shaded area in Figure 2-1, could span from Web servers to application servers to databases, with middleware serving as the glue to connect them via some sort of network. Each area is potentially susceptible to vulnerabilities. Unfortunately, because each is distinct, no single tool is capable of comprehensively assessing all platforms. Therefore, the concept of a vulnerability assessment (VA) toolkit has evolved to provide a comprehensive means of assessing vulnerabilities across the developer's continuum.

The VA toolkit, in its current incarnation, is composed of four principal tool categories:

- Web Application Assessment Tools
- Database Assessment Tools
- General-Purpose Assessment Tools
- Developer Assessment Tools.

A brief description of each tool category follows.

2.1.1 Web Application Assessment Tools

Web application assessment tools are designed to automatically scan Web applications, sites, and servers, looking for potential vulnerabilities. These tools differ from general-purpose VA tools in that they do not perform a broad range of checks on a myriad of software and hardware (i.e., port scanning or host vulnerability scanning). Instead, they perform other checks, such as potential field manipulation and cookie poisoning, which allows a more focused assessment of Web applications by exposing vulnerabilities that standard VA tools cannot detect.

2.1.2 Database Assessment Tools

Database assessment tools are vulnerability scanners designed to specifically evaluate and assess database vulnerabilities. These tools perform penetration testing and auditing, scanning for known configuration vulnerabilities, incorrect settings, weak security profiles, and missing patches or out-of-date software.

2.1.3 General-Purpose Assessment Tools

General-purpose assessment tools are the traditional security scanning tools that concentrate on scanning networks and systems for potential security weaknesses and recommend fixes. Because these tools are designed to scan a broad range of applications and systems, their ability to focus in-depth on the vulnerabilities of a specific item, such as a database, is limited.

2.1.4 Developer Assessment Tools

Developer assessment tools are designed to directly aid the application developer or software engineer. These tools are designed to locate potential vulnerabilities in either source-code or compiled programs. They do not definitively find bugs; rather they provide a reasonable starting point for performing manual security audits. These tools are mainly composed of source-code scanners, which generally are relatively new (and sometimes still under development) open-source tools designed to scan source code, finding potentially dangerous function calls. These tools will highlight potential vulnerabilities and allow the developer to either repair them or accept their risk. These tools provide better information and better prioritization than simply searching the source code for risky functions using the “grep” search tool.

Note that these tools are intended for use by the developer or someone with extensive knowledge and understanding of the source code of applications. Although these tools are not inherently complex, they require sophistication on the part of the user and intimate knowledge of the analyzed code to interpret the results.

2.2 TOOL OVERVIEW

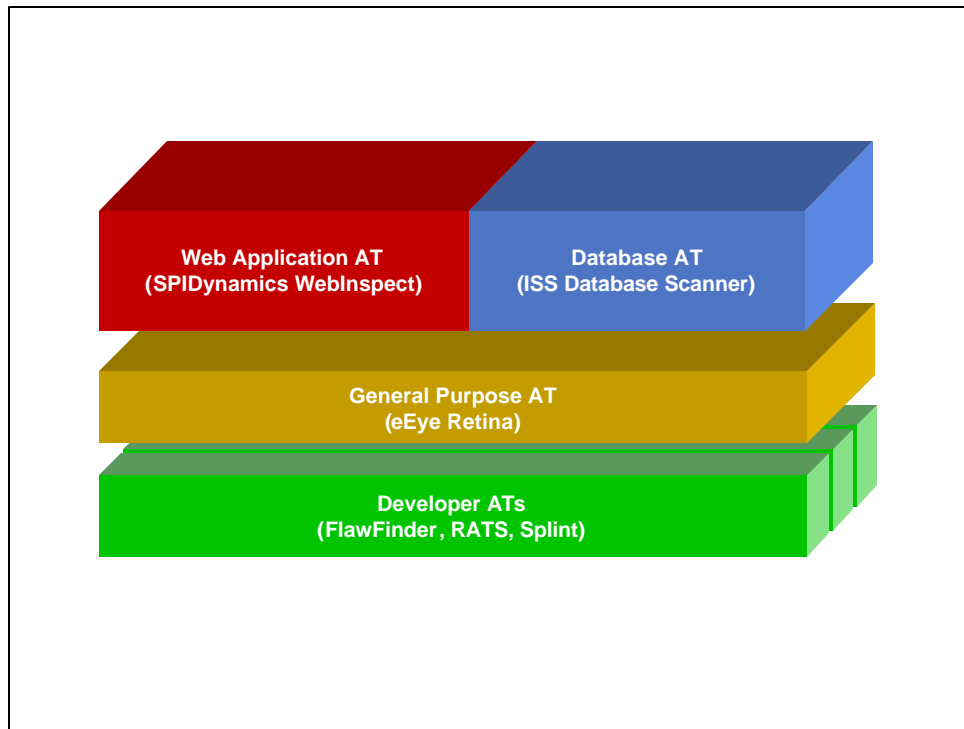


Figure 2-2. Toolkit Components

The Application Security Assessment Toolkit currently includes the four types of tools discussed in Section 2.1 and illustrated in Figure 2-2 above. Specifically, these tools by type are as follows:

- SPIDynamics WebInspect (Web application assessment tool)
- Internet Security Systems (ISS) Database Scanner (database assessment tool)
- eEye Retina (general purpose assessment tool)
- Developer assessment tools
 - Flawfinder
 - Rough Auditing Tool for Security (RATS)
 - Splint.

Note that the three developer assessment tools (Flawfinder, RATS, and Splint) possess similar capabilities across a broad range of programming languages while differing slightly in the vulnerabilities that each detects. It is therefore envisioned that the developer will use as many of the developer assessment tools as feasible to maximize the type of vulnerabilities detected.

2.2.1 SPIDynamics WebInspect (Web Application Assessment Tool)

WebInspect is designed to dynamically scan standard and proprietary Web applications to identify known application vulnerabilities. It allows the user to conduct security assessments on any Web-enabled application, including specific assessment capabilities for IBM WebSphere, Lotus Domino, Oracle Application Servers, and MacroMedia ColdFusion. Configuring the system to analyze Internet applications is easily accomplished quickly, although this can vary depending on the number of systems to be analyzed. WebInspect uses “Adaptive-Agent” technology, a set of heuristics that enables it to apply intelligent application-level security checks. This technology is a multiphased approach to Web application assessments. As a user initiates an assessment, WebInspect assigns assessment agents to dynamically catalog all areas of a Web application. As these agents complete the assessment, findings are reported back to a main security engine that analyzes the results. WebInspect then launches “threat agents” to evaluate the gathered information and apply attack algorithms to determine what vulnerabilities exist and the severity of those vulnerabilities.

WebInspect provides a full programming language and programming tools to write custom rules. The tool provides advanced executive-level reporting functionality with additional report and graphing features, including trend analysis, for comparing assessments and tracking progress. Finally, WebInspect electronically updates its built-in intelligence as it references and synchronizes (in a live-update manner) with a continuously updated database of hacking methodologies over the Internet.

2.2.2 ISS Database Scanner (Database Assessment Tool)

The ISS Database Scanner assessment tool identifies security vulnerabilities in leading database applications, such as Microsoft SQL Server, Oracle, and/or Sybase database servers. Database Scanner offers security policy generation and reporting functionality, which measures policy compliance and automates the process of securing critical online data. Database Scanner can perform two types of database scans: an audit scan and a penetration test.

The audit scan is an inside-out approach that enumerates users, groups, privileges, logins, and a wide number of other objects in the database, identifying misconfigured privileges and opportunities for misuse by authorized users. It allows users to identify exactly what objects are in their database, who has access to them, and what they have been doing and could do. Penetration testing, on the other hand, is an outside-in approach that attempts to gain access to a database the way a hacker would by using known default passwords and password guessing.

Database Scanner includes a set of predefined, customizable “best practice” security policies. These policies allow for a broad range of tests from analyzing the risk of compromise from simple attacks by

unsophisticated external attackers to testing the integrity of application data against accidental or malicious changes.¹

Database Scanner assists in identifying the following types of vulnerabilities:

- **Authentication checks** encompass all of the settings needed to verify each user's claimed identity within the database management system. Includes password strength analysis, password aging, login attacks, stale logins, default login and password checks, and security of administrative accounts.
- **Authorization checks** focus on how an authenticated user is permitted to use specific resources within the system. Includes logon-hours violations, account and role permissions, stored procedure access, unauthorized object owners, resource access, and permissions.
- **System integrity checks** focus on the coordination and control of system resources of the database system. Includes Trojan horses, operating system integrity, audit configuration, and analysis.

Database Scanner also provides executive-level reporting functionality with additional report and graphing features. Finally, Database Scanner can be electronically updated via the using its XpressUpdate feature, which provides updates to its database with the latest vulnerabilities and refined checks.

2.2.3 eEye Digital Security Retina (General-Purpose Assessment Tool)

Retina is able to scan all types of operating systems for vulnerabilities, including UNIX-based operating systems (e.g., Solaris, Linux, BSD, etc.) as well as networked devices (such as routers and firewalls). Retina includes vulnerability auditing modules for the following services: Network Basic Input/Output System (NetBIOS), HyperText Transfer Protocol (HTTP), Common Gateway Interface (CGI), File Transfer Protocol (FTP), Domain Name Service (DNS), Post Office Protocol 3 (POP3), Simple Mail Transfer Protocol (SMTP), Lightweight Directory Access Protocol (LDAP), Transmission Control Protocol/Internet Protocol (TCP/IP), and User Datagram Protocol (UDP). Additionally, Retina has modules for checking registry settings, Denial of Service (DoS) vulnerabilities, Users and Accounts, password vulnerabilities, and publishing extensions.

Retina is designed to scan for vulnerabilities quickly, concisely, and accurately and uses an artificial intelligence (AI) module, which analyzes all data gathered to determine exactly what protocol and

¹ It is recommended to use Database Scanner in conjunction with other tools from the Toolkit, such as a Web application assessment tool and a general-purpose assessment tool. The Web application tool could provide a detailed assessment of Web-enabled database front-end (if used), whereas the general-purpose tool would provide a detailed assessment and recommendations for locking down the base operating system and network services on which database servers and related critical systems reside.

service is running before proceeding and running the appropriate vulnerability checks. To find possible unknown vulnerabilities, Retina uses Common Hacking Attack Methods (CHAM), along with the AI technology, to identify potentially unknown vulnerabilities. (Note that Retina's CHAM feature has allowed eEye to discover some of the more recent large vulnerabilities in Microsoft IIS.)

Retina's use of the AI module to perform smart scanning allows for a scanning tool that is extremely fast. These features are easily and intuitively accessed through a well-designed graphical user interface (GUI) interface. Furthermore, all aspects of the scanning and reporting are easily controlled through this GUI.

Retina generates detailed information about any vulnerabilities discovered, including detailed fix/repair information. Note that in addition to including detailed fix information for all of the above known systems and service vulnerabilities, Retina can also automatically fix many of the above listed vulnerabilities, such as with registry settings and file permissions.²

Finally, as new vulnerabilities are discovered, an autoupdate feature provides updates for its modules on a regular basis. For individuals who discover their own vulnerabilities or desire other modules to be checked, Retina also has an open architecture that allows the user to create their own modules with any programming language, such as Perl, C, C++, Visual Basic, and Delphi.

2.2.4 Flawfinder (Developer Assessment Tool)

Developed by David Wheeler, Flawfinder is a Python language program that can be used to assist auditing C and C++ code. Flawfinder works by using a built-in database of C/C++ functions with well-known problems, such as the following:

- Buffer Overflow Risks (e.g., strcpy, strcat, gets, sprintf, and the scanf family)
- Format String Problems (e.g., [v][f]printf, [v]snprintf, and syslog)
- Race Conditions (e.g., access, chown, chgrp, chmod, tmpfile, tmpnam, tempnam, and mktemp)
- Potential Shell Metacharacter Dangers (e.g., most of the exec family, system, popen)
- Poor Random Number Acquisition (e.g., random).

Flawfinder produces a list of potential security flaws sorted by risk. This risk level depends not only on the function, but also on the values of the parameters of the function. For example, constant strings are often less risky than fully variable strings in many contexts. In some cases, Flawfinder may be able to determine that a construct containing a risky function is actually secure in the context of the program,

² Note that use of Retina's auto fix features should be performed with care. It is important to ascertain that any repairs made by Retina are not contrary to recommended system configurations based on DISA's Secure Technical Implementation Guides (STIG), which take priority.

thus reducing false positives. Also, Flawfinder correctly ignores text inside comments and strings (except for Flawfinder directives).

Flawfinder is quite fast, covering thousands of lines of C code on a typical desktop machine in a matter of seconds. Flawfinder is released under GNU Public License (GPL) version 2 and therefore is free software. Note that not every potential coding error that Flawfinder detects is actually a security vulnerability, and not every security vulnerability is necessarily found. In fact, Flawfinder does not “understand” the semantics of the code at all; it does only simple text pattern matching. Nevertheless, Flawfinder can be a very useful aid in finding and removing security vulnerabilities.

2.2.5 RATS (Developer Assessment Tool)

RATS, developed by Secure Software Solutions, is a security auditing utility for C, C++, Python, Perl, and PHP code. Similar to Flawfinder, RATS scans source code, finding potentially dangerous function calls susceptible to buffer overflows and Time of Check, Time of Use (TOCTOU) race conditions.³ The goal of the RATS project is not to definitively find bugs, but to provide a reasonable starting point for performing manual security audits. As its name implies, the tool performs only a rough analysis of source code. It will not find every vulnerability and will also find problems that are not vulnerabilities, such as coding mistakes.

RATS is released under version 2 of the GPL. The tool is fairly flexible in terms of defining the type of vulnerabilities it uncovers. The user can specify what vulnerabilities are reported in the tool’s output report via both the data contained in the vulnerability databases that are used and the specified setting of the tool’s vulnerability warning level. Each vulnerability includes a list of files and location information (line number), followed by a brief description of the vulnerability and suggested action.

2.2.6 Splint (Developer Assessment Tool)

Splint is a tool for statically checking C (but not C++) programs for security vulnerabilities and coding mistakes. Splint does many of the traditional Lint⁴ checks, including unused declarations, type inconsistencies, use before definition, unreachable code, ignored return values, execution paths with no return, likely infinite loops, and fall-through cases. More powerful checks are made possible by additional information given in source code annotations. (Annotations are stylized comments that document assumptions about functions, variables, parameters, and types.) In addition to the checks specifically enabled by annotations, many of the traditional Lint checks are improved by exploiting this

³ Note, however, that although RATS and Flawfinder search for similar vulnerabilities, each does so in a different manner, emphasizing different items and producing different output. Use of both tools is recommended.

⁴ Lint is a UNIX command that examines C source programs, detecting a number of bugs and obscurities. It enforces the type rules of C more strictly than the C compilers. Lint is also used to enforce a number of portability restrictions involved in moving programs between different machines and/or operating systems.

additional information. (Note that before 2002, Splint was known as LCLint. Splint 3.0 is the successor to LCLint 2.5.)

The following types of problems are detected by Splint:

- Buffer overflow vulnerabilities
- Dangerous macro implementations or invocations
- Dereferencing a possible null pointer
- Using possibly undefined storage or returning storage that is not properly defined
- Type mismatches, with greater precision and flexibility than provided by C compilers
- Violations of information hiding
- Memory management errors, including uses of dangling references and memory leaks
- Dangerous aliasing
- Modifications and global variable uses that are inconsistent with specified interfaces
- Problematic control flow, such as likely infinite loops, fall through cases or incomplete switches, and suspicious statements
- Violations of customized naming conventions.

Splint is well supported by the Secure Programming Group at the University of Virginia and comes with extensive documentation on its use. It was written in C; therefore, anyone with a standard American National Standards Institute (ANSI) C compiler can recompile it for their platform. Splint is a versatile tool because it can analyze code at many levels of abstraction.

2.3 RUNNING TOOLS FROM TOOLKIT

All of the above tools will be available in the Application Security Assessment Toolkit for use in assessing vulnerabilities within the applications under development. Depending on what stage of development (e.g., application source coding, middleware coding, system integration) or type of application under development (e.g., Web, database), the developer will be able to choose a tool to help assess vulnerabilities. The developer will be able to pick, choose, or pull tools from the Toolkit that most appropriately fit the requirements and type of application under development.

3. ASSESSMENT PROCESS OVERVIEW

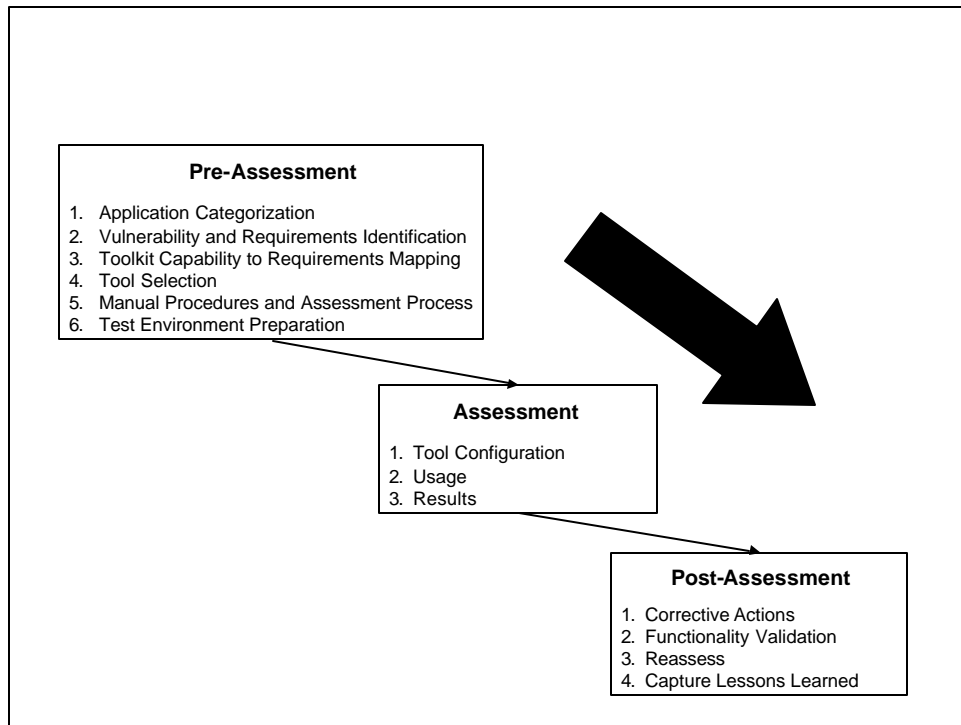


Figure 3-1. Assessment Methodology

The assessment methodology using the current Toolkit to validate an application's compliance with security requirements can be broken into three distinct phases (Figure 3-1) that are outlined below and discussed at length in the following sections.

Pre-Assessment: This phase sets the stage for the overall validation exercise. The developer will categorize the application undergoing assessment; identify security requirements that pertain to that application type; map those requirements to the capabilities of the current toolkit; select the appropriate tool(s) to conduct the assessment; develop any manual procedures to supplement any Toolkit shortfalls; and prepare the test environment. (Section 4)

Assessment: In this phase, the actual assessment of the application occurs through the use of assessment tools from within the Toolkit. Prior to the tool(s) being employed, it must be properly configured. After its use, the tool(s) will produce an output report detailing any vulnerabilities discovered, which must be reviewed and interpreted by the developer. (Section 5)

Post-Assessment: The final phase of the methodology requires the developer to take corrective actions to repair or mitigate any vulnerabilities discovered during the assessment phase. The developer must then ascertain whether the application still functions correctly, as designed, without any unintended side effects. If corrective actions were taken, the developer must repeat the assessment process to

validate that the application is free from vulnerabilities and in compliance with security requirements. Finally, the developer must capture any lessons learned. (Section 6)

4. PRE-ASSESSMENT PHASE

The methodology in this phase is an important step in the overall assessment process. Prior to the actual vulnerability assessment occurring, the developer must have a clear understanding of the Toolkit's capabilities with respect to the security requirements that pertain to the application being assessed. The pre-assessment methodology, as illustrated in Figure 4-1, will prepare the developer to successfully use the Toolkit as envisioned by helping the developer to

- Categorize the application under development
- Identify the security requirements that must be validated and any vulnerabilities to be detected
- Map the security requirements to current Toolkit capabilities
- Choose the appropriate assessment tool(s)
- Determine if supplemental procedures need to be developed to validate requirements
- Prepare the test environment.

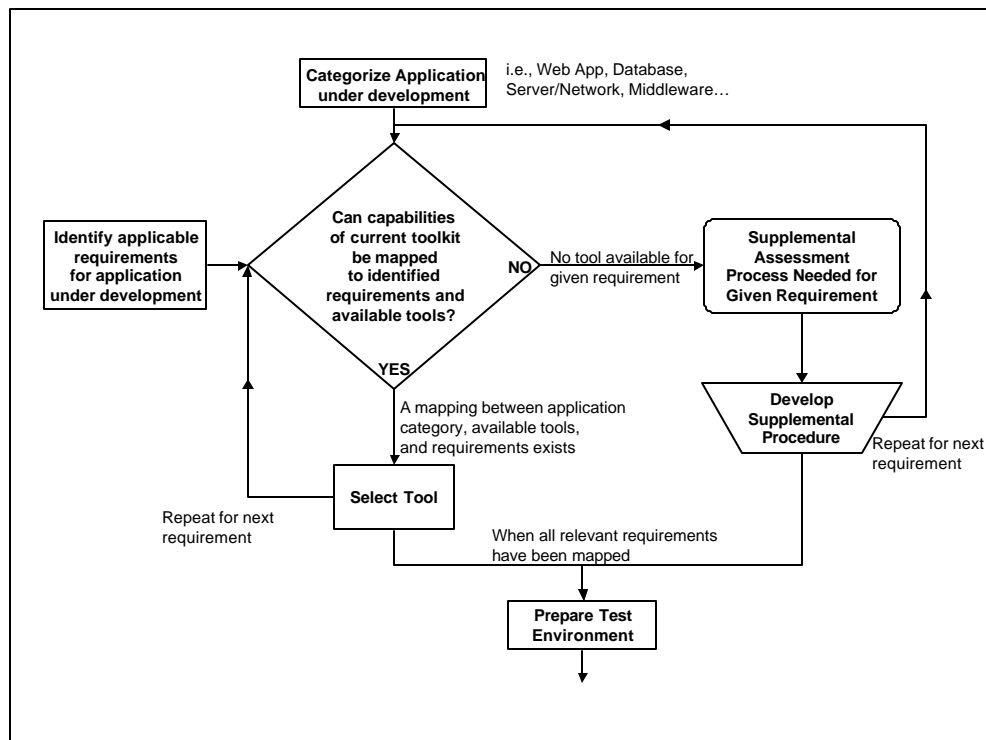


Figure 4-1. Pre-Assessment Flowchart

4.1 APPLICATION CATEGORIZATION

Application categorization, although a seemingly irrelevant and extraneous step in the development process, is important when using the Toolkit to assess application vulnerabilities. Before commencing a vulnerability assessment, the developer, by definition, will have a clear understanding of the type of application being developed. This could range from a database to a Web application, from a database server to a Web server, from a network to middleware, or a simple stand-alone application. Although this categorization will be trivial to the developer, it is an important step in the vulnerability assessment process because each Toolkit component is designed for a specific type of application. Table 4-1 lists the types of applications that can be assessed using the current capabilities of the Toolkit.

Table 4-1. Application Types

Application Type
Web Application <ul style="list-style-type: none"> • IBM Websphere • Macromedia ColdFusion • Macromedia JRun • Lotus Domino • Oracle Application Server • BEA WebLogic • Jakarta Tomcat
Web Server
Database Instance <ul style="list-style-type: none"> • Oracle • SQL Server • Sybase
Database Server
Operating Systems
Network Devices
Network Services
Host Computer
C Source Code
C++ Source Code
Perl Source Code
Python Source Code
PHP Source Code
HTML Source

Before moving on to the next step, the developer must categorize the application being assessed. Categorizing the application will become important during the Toolkit Capability to Requirements Mapping (4.4) when the developer must map the most appropriate tool to the application. In some instances, a requirements mapping may exist for one type of application (i.e., a Web application) but not

for another (i.e., database). The developer therefore must have a clear idea of the application category to be able to correctly determine if a mapping does exist. The use of multiple tools may also be appropriate when analyzing an application that falls into several categories (e.g., a Web application with a database interface), or when analyzing actual source code in which case multiple developer assessment tools may be used.

4.2 VULNERABILITY AND REQUIREMENTS IDENTIFICATION

The *Recommended Standard Application Security Requirements* document (Version 1.1, June 6, 2002) defines and documents a set of recommended security requirements that is common to all software applications. That document serves as the first step in designing security into applications and will aid application developers in identifying potential vulnerabilities and security flaws early in the life cycle of the application.

After the developer has categorized the application currently under development, the developer will need to identify the relevant security requirements that pertain to the application using the *Recommended Standard Application Security Requirements* document.⁵ Before commencing the next step, the developer should have assembled a complete list of pertinent security requirements for the application that is being assessed. This list will be used in the next step to determine which tools within the Toolkit can be applied for the current assessment.

4.3 TOOLKIT CAPABILITY TO REQUIREMENTS MAPPING

The developer will proceed to use the list of pertinent security requirements to determine which tool within the Toolkit (if any) can be used in assessing vulnerabilities within the current application under development. The developer will use the Requirements-Toolkit Capability Map (see Appendix C) to determine which tools can be used to validate the security previously identified requirements and assess vulnerabilities within the application. If a security requirement is mapped to an appropriate and available assessment tool, then the developer can select the tool and be able to assess that requirement in an automated manner (Section 5). However, if a direct mapping does not exist, then the developer will have to validate the requirement using supplemental procedures and processes (see Section 4.5).

Consider the example where the developer's application is a Web application. The developer identified three requirements pertaining to this application as depicted in Table 4-2. The developer then cross-references the Requirements-Toolkit Capability Map to identify the tool to run.

⁵ Note that use of the *Recommended Standard Application Security Requirements* guide is covered within the guide itself but not in this Methodology document. Familiarity with the *Recommended Standard Application Security Requirements* guide is assumed.

Table 4-2. Example of Successful Toolkit to Requirements Mapping

REQ #	QUICK REFERENCE	REQUIREMENT	VULN. ADR.	TOOL CATEGORY MAPPING
4.0.10	High-risk services	Avoid use of high-risk services and technologies, such as Telnet, Simple Network Mail Protocol (SNMP), and mobile code, in/by applications unless absolutely necessary.		<ul style="list-style-type: none"> • Web Application • General Purpose
4.2.17	Invalid pathname references	Whenever a pathname or Uniform Resource Locator (URL) referenced in the application code is changed or removed from the system, the application code must be changed to change or delete that reference.	V14	<ul style="list-style-type: none"> • Web Application
4.4.18	Initialization of variables	<p><i>If the programming language in which the application is written does not automatically ensure that all variables are initialized to zero when declared:</i></p> <p>The application code must explicitly initialize all of its variables when they are declared.</p> <p><i>NOTE: “C” does not provide the necessary zero initialization of variables.</i></p>		<ul style="list-style-type: none"> • Web Application • Developer

4.4 TOOL SELECTION

After completing the Toolkit capability-to-requirements mapping, the developer will have identified a set of assessment tools that can be used to validate requirements within the application. If multiple tools were identified as appropriate, the precise order of their use is left to the developer’s discretion. For instance, the mapping in Table 4-2 may have identified a Web application tool and a general-purpose tool as appropriate tools for testing a Web server and Web page. The developer is then ready to prepare the test environment and begin the assessment process. If no tools were identified, then the developer must resort to supplemental procedures to validate any identified security requirements for the application under development.

4.5 SUPPLEMENTAL PROCEDURES AND ASSESSMENT PROCESS

Supplemental procedures and assessment processes will be required if no direct mapping exists between a requirement and toolkit capability. This may occur when an application is highly customized for a specific task (and therefore beyond the typical scope of a commercially available tool). In addition, an application may be a “legacy” application operating on hardware (or software) that is not widely supported in the environment in which many of the commercially available tools were developed. Finally, many of the current security requirements contend with either classified data or the use of encryption—typically areas where many commercially available tools avoid. For example, consider the mappings in Table 4-3 that are based on the current Toolkit.

Table 4-3. Example of Requirements Needing Supplemental Procedures for Assessment

REQ #	QUICK REFERENCE	REQUIREMENT	VULN. ADR.	TOOL CATEGORY MAPPING
4.0.6	Interoperability with Department of Defense (DoD) Public Key Infrastructure (PKI)	The application's PKI functions must be implemented using PKI technology that is interoperable with DoD PKI. [7]	V3	None. Use Supplemental Procedures
4.2.14	Labeling of Classified data	<p><i>If the application is either</i></p> <ul style="list-style-type: none"> <i>Used to create or modify classified data, OR</i> <i>Is a Mission Category I used to create or modify classified or sensitive data.</i> <p>The application must apply the appropriate confidentiality and integrity labels to the data at the time of creation or modification. These labels must be understood by the access control mechanism used to control access to the data.</p>		None. Use Supplemental Procedures

In the "Tool Category Mapping" column, no direct mapping exists between the requirement and a tool (indicated by the "None. Use Supplemental Procedures" statement). Therefore, for the application developer to validate these security requirements, manual procedures or processes will need to be developed as the current Toolkit does not contain any capability to allow for an automated assessment to occur.

To successfully validate these types of requirements, the developer will need to devise a method to manually test the application. As a first step, the developer should refer to the *Application Security Developer's Guide*. The Developer's Guide was designed to provide guidance and recommendations to developers interested in securing their applications and as such will provide insight to the developer in creating a custom methodology for validating requirements that are beyond the capabilities of the toolkit. At this point, the developer has several options available to validate the requirement. For example, the developer may—

- Determine that he/she possesses sufficient knowledge on the requirements on hand and will devise a way to validate them.
- Examine current Security Technical Implementation Guides (STIG) for relevant procedures for securing the application to meet the requirement.
- Develop a customized script (e.g., a Perl or Python script) to automate the testing process.

Note that occasionally an indirect mapping will occur with regard to a certain requirement for the current Toolkit. This occurrence, in which an automated tool will provide a means of indirectly validating a security requirement, is indicated in the right-most column of Appendix C by the phrase, "Indirect

mapping using tool name(s) where tool refers to the tools that can be used indirectly to validate the requirements.

Consider the example in Table 4-4 below.

Table 4-4. Example of Indirect Toolkit to Requirements Mapping

REQ #	QUICK REFERENCE	REQUIREMENT	VULN. ADR.	TOOL CATEGORY MAPPING
4.6.5	Audit information captured by classified applications	<p><i>If the application handles classified data:</i></p> <p>Each audit record must include the following information (as relevant for the type of event) [1,2,18]:</p> <ul style="list-style-type: none"> • UserID of user or process ID of process causing the event • Success or failure of attempt to access a security file • Date and time of the event • Type of event • Success or failure of event • Seriousness of event (violation)* • Successful or failure of login attempt • Denial of access resulting from excessive number of login attempts • Blocking or blacklisting a UserID, terminal, or access port, and the reason for the action • Data required to audit the possible use of covert channel mechanisms • Privileged activities and other system level access • Starting and ending time for access to the application • Activities that might modify, bypass, or negate safeguards controlled by the system • Security-relevant actions associated with periods processing, or the changing of security labels or categories of information 		Indirect mapping using database, Web application, and general-purpose tools

The database, Web application, and general-purpose tools, can all be used to indirectly validate Requirement 4.6.5. Running these tools will deposit traces of their behavior in audit logs. This will only occur, however, if auditing is correctly enabled on an application. The developer can use these three tools to generate audit logs and then manually test them to validate the requirement by analyzing what behaviors were recorded. Thus, indirect mappings can be considered a hybrid between a supplemental process and a capability provided by an automated tool within the Toolkit.

It is imperative that all security requirements be validated, regardless of the methods chosen by the developer. As the Toolkit evolves with time, it will be expanded to include more tools capable of validating more requirements in an automated manner, thereby easing the burden placed on the developer to create a manual process or procedure. However, until all requirements for all types of applications are mapped to an automated tool, the developer will have to rely on manual assessment.

4.6 TEST ENVIRONMENT PREPARATION

The Application Security Assessment Toolkit should be employed in an isolated or closed network test environment to shield an organization's production networks from the assessment tools. Many of the tools employ scanning techniques that are designed to probe and disable their targets while searching for vulnerabilities. These techniques will activate intrusion detection devices within a network indicating that an internal attack on the network was under way, unnecessarily alarming network and security administrators.

To circumvent this problem, the Application Security Assessment Toolkit should be installed on a

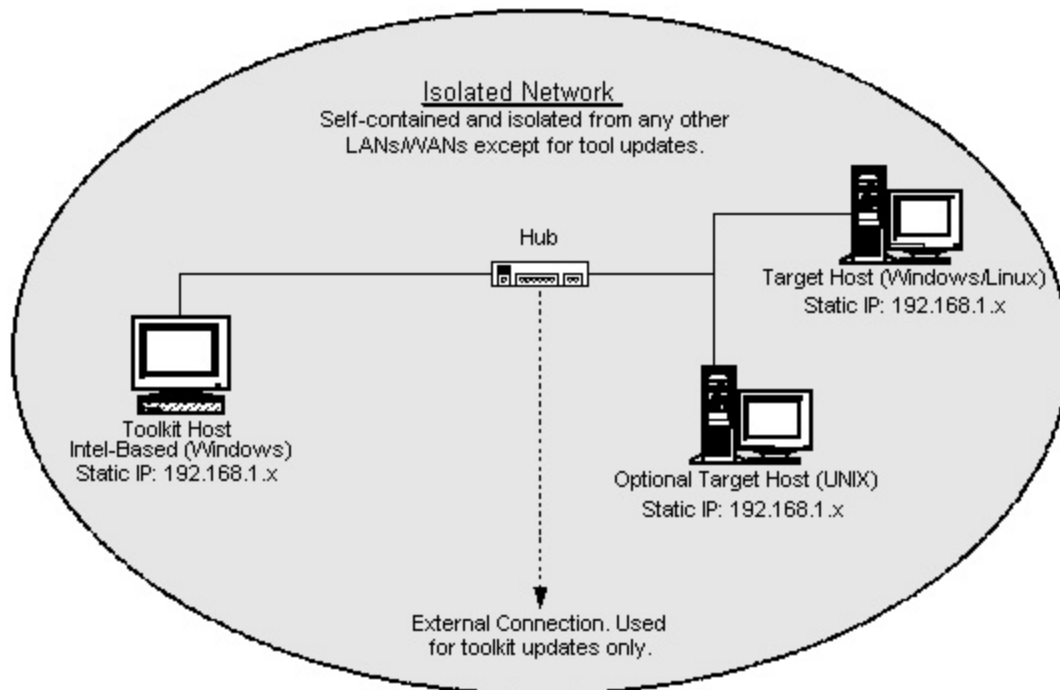


Figure 4-2. Isolated Test Network

closed and isolated test network. Figure 4-2 illustrates a potential network layout.

To test an application for vulnerabilities and compliance with security requirements, the developer will need to install the application on one of a set of target host computers within the test network. Typically, there will be several target hosts varying in operating system and hardware, each with its own statically defined IP address. These target hosts will be connected via a network device, such as a network hub,

to form a simple network. An additional Intel-based (Windows operating system) computer will host the Toolkit and will also be connected to the network.

The developer will need to install the application under test on the appropriately configured target host. Then, using the Toolkit host, the developer should proceed to test for vulnerabilities. Note that because of the myriad types of applications that can be built and tested, the exact details of configuring the target host will be left up to the developer.

5. ASSESSMENT PHASE

In this phase, the user of the tools will configure and run the tools identified in the pre-assessment phase and then interpret the results:

- **Tool Configuration.** The developer will configure connections between tool and the application, as well as any internal settings within the tool.
- **Tool Use.** Once configured, the tool will be explicitly started (via a GUI switch/button or from the command line) to perform its operation automatically.
- **Tool Output.** Once the tool's scan is complete, the tool will typically output its results into some form of report or file (which can be configured for verbosity). The developer will read and interpret this output file to determine what, if any, vulnerabilities exist within the application.

The process for configuring, using, and interpreting each tool within the current Toolkit (as described in Section 2.2) is outlined below.

5.1 SPIDYNAMICS WEBINSPECT

This discussion is based on WebInspect version 2.6.8 and is intended to be a supplement to the WebInspect User's Guide.

5.1.1 Tool Configuration

The developer will configure the tool for use in a development environment (nonproduction) as this tool can have detrimental affects on a Web application during the process of testing.

5.1.1.1 Smart Update. The Smart Update feature allows the user to download the latest vulnerability, adaptive agents, and product updates from SPI Dynamics to ensure that the tool is functioning with the latest technology. The Smart Update feature should be used before each scan is initiated to ensure the latest vulnerabilities are being checked for. (Note that if product updates are available, they may be skipped and installed later.) If the vulnerability database becomes corrupted, the Refresh Database feature may be used to refresh and restore your database.

5.1.1.2 Target Selection. The selection of the target is made via the Uniform Resource Locator (URL) field. The target selection can be in the form of either a complete URL or an Internet Protocol (IP) address. If only certain portions of the Web site need to be tested, a starting directory root can be appended to the end of the supplied URL or IP address.

5.1.1.3 Form Input. If the Web application being tested contains pages with submittable forms, WebInspect is capable of automatically filling in and submitting the forms. WebInspect will fill in the forms automatically with data provided from the text file “formvalues.txt”.

In the formvalues text file, comment lines begin with the semicolon (;). The fields used to fill the forms are one per line, with three columns on each line. The first column is the name to search when identifying forms to fill. If the name to search for is preceded by an exclamation point (!), then only exact matches will have their form filled. The second column contains the data that will be inserted when a form with the name from the first column is identified. The third column contains one of three options: 0, 1, or !x where the x is a number. These three options declare the maximum amount of characters that a form field will be filled to be a match. The definitions of the options are as follows:

- **0** – ignore the maximum length
- **1** – populate the field with the maxlength of characters
- **!x** – maxlength of the field must be equal to x to be considered a match.

A shortened example file might be as follows:

```
; Comments
; Comments
first    Bob    0
!mid     E      0
last     Zmuda  0
phone    123-456-7890 !12
phone    1234567890  !10
add      123+Fake+Address+Way    0
```

This translates to the following when certain form fields are encountered:

- **first**—“Bob” will be filled into the form
- **middle**—There is no match in the example as a result of the ! before *mid*; nothing will be filled into the form
- **last**—“Zmuda” will be filled in to the form
- **phone**—If the field is of length 12, “123-456-7890” will be filled into the form
- **phone**—If the field is of length 10, “1234567890” will be filled into the form
- **phone**—If the field is of a length other than 10 or 12, nothing will be filled into the form
- **address**—This matches *add*, so “123+Fake+Address+Way” will be filled into the form.

5.1.1.4 Policy Selection/Creation. When scanning and testing, WebInspect bases its checks and tests according to a user-selectable policy. The policies available by default are Quick Scan, Safe Scan, Full Scan, Assault Scan, and Blank Scan. These policies are defined below.

- Quick Scan—includes an automated and logical link exploration (crawl) of the server in addition to performing checks for known vulnerabilities in major packages and unknown vulnerabilities at the Web server, Web application server, and Web application layers. A Quick Scan does not perform checks that are likely to create DoS conditions.
- Safe Scan—is a limited version of the Quick Scan. It includes a subset of the checks included in the Quick Scan, eliminating checks that will trigger DoS conditions on systems of any sensitivity.
- Full Scan—is an extension of the Quick Scan. It includes all the checks possible, including those for minor packages. The only checks that the Full Scan does not perform are those that are likely to create DoS conditions.
- Assault Scan—is the scan that will perform all checks, including those vulnerability checks that can potentially create DoS conditions.
- Blank Scan—is an empty scan that does not check for anything. It can be used as a starting point and allow the user to individually add each check desired. This is the recommended course of action because even though the default scans may appear safe by their respective labels, there is the possibility that specific applications could still be adversely affected by WebInspect.

Although it is easy to blindly use one of the above scan policies, it is recommended that users familiarize themselves with all the options within the policies and then create a custom policy (using the Blank Scan policy). This is recommended so that the user becomes familiar with every check and test that is to be performed and is able to identify and eliminate performing potentially dangerous checks and tests. Lastly, as a failsafe, all testing should be performed in a test environment in which the developer can revert to a previous state or quickly reload the application if it becomes corrupted by the tool.

When creating a custom scan policy, custom agents may also be added. Custom agents are actually custom scripts that can be developed from within WebInspect to test for nearly anything that WebInspect is capable of testing.

5.1.1.5 Settings. Settings for the WebInspect can be accessed from the Tools->Settings menubar.

- General
- Localhost Proxy Request and Response Rules
- 404 Page Settings

- Crawler Settings
- Attack Settings
- Proxy Settings.

The **General Settings** include:

- Connection Settings—change the length of time, in milliseconds (ms), that the tool will wait before canceling a request when there has been no response from the Web application. The default setting is for 2000 ms, or 2 seconds. This should be adjusted up (higher) if it is known that the Web application will take more time on average to respond to requests, as a result of a slow connection or slow Web application. Adjustments down may be possible to increase the speed of the scans, but only when there is confidence that this will not cause requests that are valid to time out.
- Authentication Settings—allow for Basic and NT LAN Manager for Microsoft Windows (NTLM) authentication to occur when provided with the necessary credentials.

The **Localhost Proxy Request and Response Rules** allow for WebInspect to be used as a proxy server for surfing the Web application. WebInspect will then capture the information from the Web application as you browse and click through the site. The settings for this feature are further explained in detail within the WebInspect User's Guide.

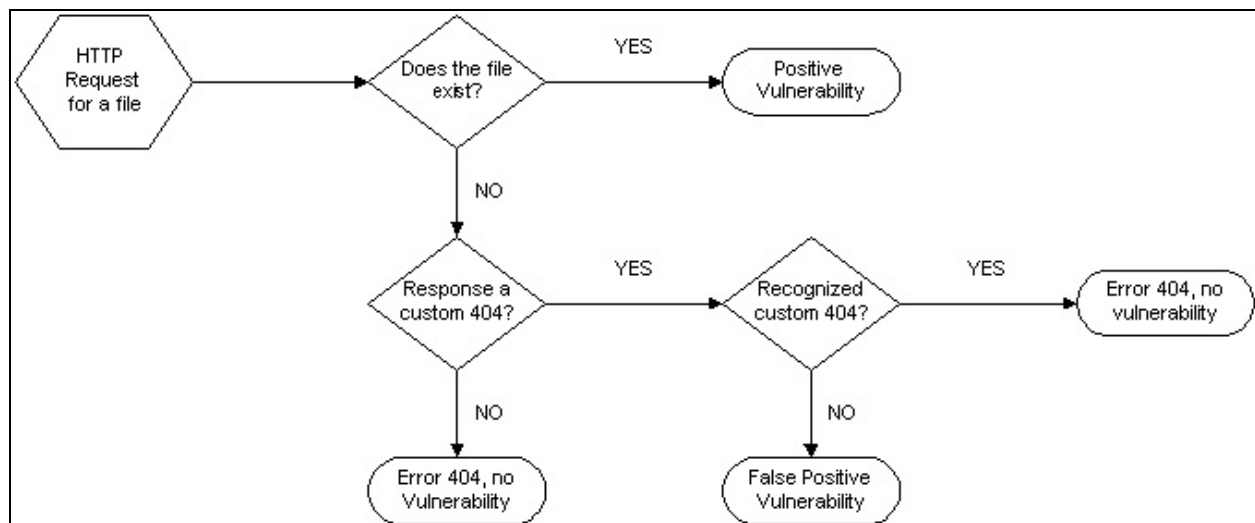


Figure 5-1. Possible 404 Page Errors

The **404 Page Setting** is one of the most important settings to configure. When browsing a Web application, anytime a file or Web site is not found, an *error 404 not found* message is returned (Figure 5-1). WebInspect uses this error code to identify whether a request has succeeded or failed. In many

cases, Web applications modify the standard error 404 page to create custom 404 pages. If WebInspect is unable to recognize these custom error 404 pages, it can cause the results of WebInspect to be full of false positives. For example, when the WebInspect requests a certain file that contains a known vulnerability, WebInspect must be able to accurately determine if the file exists or not. If a custom 404 page is returned that WebInspect does not recognize, WebInspect will record that the file was found when it actually was not. This creates a false positive, or a condition in which a vulnerability is identified when the vulnerability does not actually exist.

The **Crawler Settings** allow the user to custom configure headers and regular expressions to be included in the crawl. In addition, these settings allow the user to further specify what will or will not be crawled. The headers that can be configured are those used by the HTTP data exchanged with the Web application while conducting a crawl. By adding custom headers, such as “Crawling by WebInspect on MM/DD/YY,” all requests to the Web application will be easily identified within the Web server logs. In configuring what will or will not be scanned, Table 5-1 lists the possible Crawler Settings available.

Table 5-1. Available Crawler Settings

Setting	Description
Exclude Extensions	File Extensions that should be ignored during crawl
Allow these hosts to be crawled	Additional hosts that should be crawled
User-Defined Regular expressions for URLs	Add regular expressions to a crawl when beneficial to completing a crawl
Do Not Crawl	Hosts that should not be crawled
Include Referrer Header	Yes/no on including the referrer information on users coming from other sites
Use URL Sanity Checker	URL sanity checker to help remove requests that use javascript
Max URL Visits	Maximum number of URLs to visit during the crawl, 0 sets to unlimited
Request Timeout (sec)	Timeout period for a request to fail
Request Method	Alternative methods of requesting Web pages, such as Get, Post...
Crawl Depth	Depth into Web application tree crawl should go, 0 sets crawl depth to unlimited

The Exclude Extensions options should be reviewed carefully. Its main purpose is to avoid excessive time consumption during testing while downloading graphics and other types of files that do not cause vulnerabilities. However, if those certain types of files are required to be available for download from a Web application, then the user should ensure that the corresponding file extensions are not on the Exclude Extensions list. For example, if the user wants to ensure that the Web application does not contain any downloadable Power Point slides (.ppt), then by removing the .ppt extension from the

default Exclude Extensions option, WebInspect will be able to identify any instances in which .ppt files are available for download. In addition, certain file types may contain information that should not be disclosed, such as a PDF file (.pdf) that contains the security policy for the network the Web application resides on. Consequently, close attention should be paid to which extensions are placed on the Exclude Extensions list.

It is important to complete the Do Not Crawl list so that parts of the Web application the user does not want to test are not included in the testing. In addition, if WebInspect's license allows for any host to be scanned, it is important to accurately list allowed hosts so that WebInspect's crawling and testing does not extend to hosts not belonging to the user or outside the testing authorization. If the tools are used in a development environment as recommended, this can be further ensured by testing on a closed network containing only the testing computer and the Web application host computers.

To obtain a complete crawl and test of the Web application, the Max URL Visits and Crawl Depth options should be set to **0**. This sets the fields to unlimited so everything with a link to it should be crawled and identified for testing purposes.

The **Attack Settings** contain an area in which to add header information to all requests that are used during the attack phases of the tool use. This is similar to what is done with the Crawler Settings during a crawl. By adding custom headers, such as "Attack by WebInspect on MM/DD/YY," all requests to the Web application during the attack phase will be easily identified within the Web server logs.

The **Proxy Settings** can be used to set up any proxy information needed when a proxy is required to browse to the Web application being tested. In addition, any Client Certificates required can be set up with this option.

5.1.2 Tool Use

5.1.2.1 Modes. After configuring the tool and beginning the process to start a scan, the following four modes of scanning possible:

- Scan
- Crawl
- Step Mode
- Open Saved Scan.

The **Scan** mode will fully crawl and map out the Web application's tree and then apply all attack methodologies to the site. All attack methodologies that are to be applied are determined by the policy that is selected for use.

The **Crawl** mode is used to only fully crawl and map out the Web application's tree. Attack methodologies will not be applied to the site until the Audit button is pressed.

The **Step Mode** allows for the user to set WebInspect up as a proxy and to record your actions as you navigate the Web application. This allows the user to manually map out the sections that are desired to be included in the audit. Attack methodologies can then be applied to the sections manually crawled by pressing the Audit button.

The **Open Saved Scan** mode allows the user to open previously saved scans for further evaluation, crawling, or auditing.

5.1.2.2 Usage Flowchart. This section describes an overall strategy to using WebInspect. When performing a test, the following sequence of steps should be followed, looping back as necessary (see Figure 5-2):

- Smart Update
- Crawl
- Manual Crawl
- Audit
- Information and Reporting
- Patching, Fixing, and Repairing.

Smart Update

The first step to starting any scan is to run the Smart Update. As was noted earlier, it is important to ensure that the tool is operating with the most up-to-date techniques for discovering and identifying vulnerabilities in the Web application

Crawl

The second step to starting the scan is to run a crawl. The crawl will identify and map the entire structure of your Web application. From within the scan wizard, select the Crawl setting, the URL or IP address to scan, and the box to allow submission of forms during the crawl if desired. When allowing forms to be submitted, make sure to configure the *formvalues.txt* file with appropriate data to fill in the fields that may be encountered while crawling the Web application.

Once the type of crawl is selected and the form values are determined, the policy to use for the scan needs to

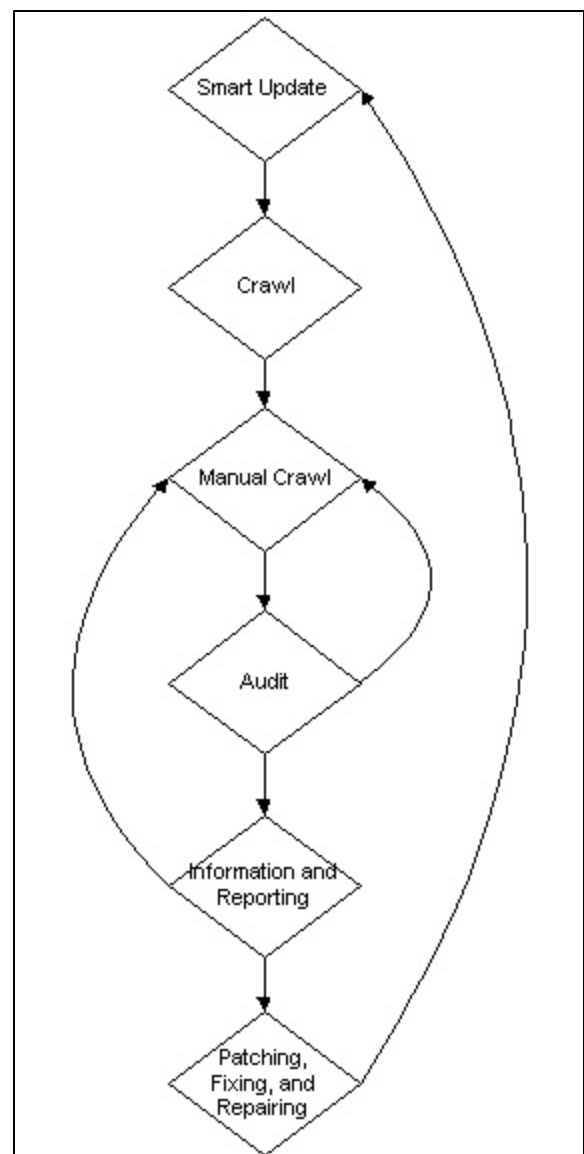


Figure 5-2. WebInspect Usage Flowchart

be selected. The policy selected here should be the custom policy that has been developed specifically for testing the Web application. If in a development environment and the appropriate back-ups of the Web application have been made, the Assault policy may be used to fully test the Web application. If it is found that when the Audit occurs there are problems completing the Audit, then adjustments may be needed to reduce the checks of the Assault policy down to a safer set. Once these adjustments are made, the new custom policy should be selected and used in the future.

Manual Crawl

After the crawl has completed, the findings and tree structure of the Web application should be reviewed and edited as necessary. If any directories are not found that should have been included in the test, then these directories should be manually added to the tree structure. If it is found that any directories were found that are not to be included in the testing, then these should be removed.

In the site map of the Web application, right clicking with the mouse on any directory or file within the tree structure will bring up a menu of options. Table 5-2 lists these options.

Table 5-2. Manual Crawl Options

Menu Item	Description
Copy	Copy the URL of the directory or file
Edit Session	Edit the request within the Request Editor to make new custom requests
Remove Children	Remove any files or directories beneath the current directory selected
Manual Step-Thru	Use WebInspect as a proxy to manually step through and add directories or files to the tree structure
Crawl	Start a new crawl from the point selected; useful if earlier crawl was restricted in scope
Crawl 1 Depth Level	Start a new crawl from the point selected, but only go a depth of one deeper
Ignore Vulnerability	If a vulnerability is identified with which you do not agree, remove from the findings
Help	Open the WebInspect help dialog box
View	View associated pieces of the tree structure

Using the above methods, the user should be able to completely map out all known pieces of the Web application quickly and accurately. In addition, the user should be able to remove any pieces of the Web application that should not be assessed further. Any parts removed should be noted and later added to the restrictions in the settings if possible to save time in future scans.

If there is not enough information in the above, the window on the right provides additional information. When a section of the site in the Site View tree structure of the Web application is selected, three tabs are available in the right window: Report, Session Audit, and Properties.

The Report tab will summarize the findings for any file or directory that is selected if there are any. The Session Audit tab will provide a means to review and modify the policy used for audit on the particular

files or directories. The Properties tab will provide a complete log of what requests were sent out and what responses were received for each directory and file that is found. The Table 5-3 lists the information that is recorded.

Table 5-3. Properties Tab Options

Command	Description
Show Request	The raw request that was sent to the Web application is recorded here
Show Response	The raw response for the above request is recorded here
Web Browser	The response for the above can be viewed within a browser here
Show Details	The details of the request can be found here, such as Response Code, Request Method, Response Time,...
Show Links	Show all links within requested page that were found
Show Comments	Show all comments within requested page that were found (“<!...”)
Show Text	Show all text that was found within requested page
Show Hidden	Show any hidden fields that were found within requested page (type=“hidden”)
Show Forms	Show any forms that were found in the requested page (“<FORM...”)
Request Editor	Open request from above in the Request Editor
Help	Open the WebInspect help dialog

Audit

Once the Manual Crawl has been completed, the user can start the audit. However, before starting, it is important to make sure any issues from the crawl have been corrected within the settings and any policy changes needed have been implemented. To start the audit, click the Audit button and wait for the audit to complete.

As the audit runs, any vulnerabilities discovered will appear in the lower window and the user is allowed to view them and create reports at any time; however, it is recommended that this not be done until the audit has been completed. Note that the time for the audit to complete will depend on the number of links identified in the crawl, the policy selected for the audit, and the timeout for requests to fail.

Information and Reporting

As described in the manual crawl process, a wealth of information exists that can be viewed from within the tool before any reports are even generated. From this point, two options are available to the user: (1) use the information found within the tool to manually crawl and or request pages from the Web application, and (2) simply run the reporting features and generate reports that summarize and list the findings. The tools capabilities, combined with a user’s expertise, can allow for multiple loops in the flowchart to occur before a user is satisfied with their findings. Because a complete listing of all the possible vulnerabilities can never be guaranteed, users must decide when they have exhausted their expertise and the features of the tool in discovering and identifying the vulnerabilities present in the Web application.

At any stage in the process, a report can be generated. There are four main types of reports available within WebInspect:

- Simple Report
- Executive Report
- Custom Report
- Trend Analysis.

The **Simple Report** is a concise summary of the vulnerabilities found, along with summary descriptions of the vulnerabilities found, how the vulnerability can be exploited when information is known, information on fixing or patching the vulnerability, and references for where additional information can be found for that particular vulnerability.

The **Executive Report** is a one-page graphical summary of what vulnerability types have been found along with some other summary graphs of the associated vulnerability information. This includes a table with the number of each type of vulnerability found (Critical, High, Medium, Low, Info), the number of each issue type found (Application, Operational, Custom Check), and the number of each type of application data that has been found (Comments, Cookies, Email, Hiddens, Scripts, Web Forms, Java Scripts). The type of each vulnerability and issue found are then placed into graphical pie charts to illustrate percentages for each.

In addition, there are two bar charts that show the types of files that were analyzed (blank, .asp, .aspx, .dll, .exe, .htw, ...) and the attack groups where vulnerabilities were found (Directory Enumeration, Web Application, Unknown Application Testing, Web Servers, WebInspect Internal).

The **Custom Report** can be broken into two separate pieces, or remain combined. The two pieces are the custom vulnerability report and the custom application report. The custom vulnerability report allows for any or all of the vulnerability classifications to be included in the report. This includes critical, high, medium, low, and info classifications of the vulnerabilities. The custom application report contains the data that is found in the Information part of Site View, or the left window of the tool. This data is shown in Table 5-4.

Table 5-4. Data Reporting Options

Application Setting	Description
Email	Report any email addresses that were found
Hiddens	Report any hidden fields that were found
Comments	Report any comments that were found
Cookies	Report any cookies that were found
Key Word Search	Perform key word searches on expressions added by user. These can be regular expressions also
Forms (Post From)	Report any forms you can post from that were found
Scripts (Post To)	Report any scripts you can post to that were found
Action	Report on any action (POST, GET, PUT, INDEX) found

Java Script	Report on any Java Scripts that were found
Response Status	Perform response status search on input by user, such as identifying pages with response codes of “403”

The **Trend Analysis** scan allows the user to compare either two previous scans, or one previous scan with the current scan. The report generated will compare the number of critical, high, medium, and low vulnerabilities between the two selected scans. It will also generate some bar graphs to highlight the changes.

Patching, Fixing, and Repairing

Once the reports are generated and the findings are reviewed, any vulnerabilities discovered should be patched, fixed, repaired, or mitigated to a state where the level of risk is considered acceptable. Because this effort will require changes in some form to the Web application, any assumed level of risk of the Web application is no longer valid. Anytime changes are made to a Web application, new and potentially higher risks are introduced. Therefore, it is very important to consider every change to be implemented before acting, log all changes that are made, and to start over from the beginning to reevaluate the risk level and vulnerabilities that may be present in the Web application. Through this continuous cycle of identifying potential vulnerabilities and repairing them, a more secure Web application can be designed. This will not ensure total security without any risk, but it will help to greatly reduce the risk level.

5.1.2.3 Extra Tools. WebInspect offers a few extra features and tools that are outlined below.

Request Editor. The Request Editor allows the user to edit requests and then resend them to the Web application. The responses can then be viewed either in raw HTML or in a browser. This is useful to modify requests that have already been sent during the scanning and testing process when reviewing the results.

Regex Tester. The Regex Tester can be used to verify regular expressions. Regular expressions are a means of generically identifying a string with the use of a pattern. An example of this would be searching for a number between 0 and 9. One method would be to individually search for each number. A regular expression to accomplish this in one search would be “[0-9]”. Expressions can be much more complex, so the Regex Tester can be used to test and verify a regular expression before use. Regular Expressions may be used in Request Rules, Response Rules, and Crawler Settings.

Web Discovery. The Web Discovery utility is a basic port scanner. This utility will allow the user to determine what ports are open on an IP address. This can be useful in identifying additional unknown ports an IP address may have available.

Encoders/Decoders. The encoders and decoders utility can be used to either encode or decode the following:

- Base64
- Hex
- URL Encode

- Unicode (Latin)
- MD5.

5.1.2.4 Command Line. WebInspect can be operated from the command line to easily define a scan to take place. To schedule a scan, a command similar to UNIX cron can be used. Scheduling a scan can be useful to automatically test changes that are made every day in the development environment before they are scheduled to be updated on the production environment. Command line execution is accomplished with the program wi.exe with the flags shown in Table 5-5.

Table 5-5. Command Line Flags

Flag	Definition
-r <filename>	Export report to <filename> (HTML report only)
-a user:password	Basic Authentication
-L	Limit scan to root URL
-h <url>	URL to scan
-policy <filename>	Policy file to be used (*.apc)
-S	Saved scan file name
-? or /?	Help
/stop	Stop WebInspect

For example, to start a scan of the example site using the assault policy and outputting a report, the following command would work:

```
wi -h http://www.example.com -r example-report.html -policy Assault.apc
```

5.1.3 Interpreting Tool Output

At any time, all the information from the scans performed by WebInspect is available to the user. This includes all requests and responses made by WebInspect during the test process. This includes the site tree structure and information found within the Site View window along with report summary information on findings from the crawl and audit from the Report tab. In addition, the Properties tab includes all of the requests and responses that were sent or received by WebInspect along with further key information from the requests and responses highlighted. Further information and elaboration about these features can be found in Section 5.1.2.2.

The main information discovered by the tool will be the vulnerabilities that are identified. These can be found in the listings of the Alerts in the bottom window of the WebInspect screen. By selecting a vulnerability from this screen, it will then be highlighted in the Site View window. If the vulnerability exists in multiple locations of the Site View, a selection box will pop up to allow for selection of which location to be seeking information from.

Once the vulnerability is highlighted within the Site View window, the Report and Properties tabs will be available in the right window. By selecting the Report tab, the right window will contain a Summary report of the item selected in the Site View window. If the finding is also a vulnerability selected from the Alerts window, then additional information may be provided, such as Execution, Fix, and References. The Summary provided will contain a brief summary as to why the finding is or is not a vulnerability. The Execution part will explain how the vulnerability can be taken advantage of if there are publicly known methods. The Fix section will provide the recommended action the user should take to patch the vulnerability. The References section will provide alternative references for additional information on the vulnerabilities found when available.

If the specific requests and responses are not needed to be reviewed by the user, then a report with the Summary, Execution, Fix, and References can be generated by the tool. In addition, other types of reports and summaries of information can be included within the reports generated. The types of reports that can be generated and the information that can be found within them are detailed further in Section 5.1.2.2.

The vulnerabilities identified by WebInspect will be placed into five categories. These five categories are critical, high, medium, low, and info. These are determined based on the risk levels that WebInspect has placed each vulnerability within. Note that the risk level that WebInspect assigns may not correspond to the risk level assigned by the developer as it pertains to his environment or security policy. Consequently, no vulnerability at any risk level should be ignored when risk mitigation strategy is being developed. Risk mitigation for an “Info” category vulnerability could be just as important as that of a “Critical” level vulnerability.

Vulnerabilities Report	
Server:	http://www.example.com:80
Severity: Critical	Since exploiting this vulnerability would involve ...
Count: 1	http://www.example.com/_vti_bin/_vti_aut/author.dll
Summary:	Since exploiting this vulnerability would involve crashing the web server, WebInspect does not perform false-positive checking for it. The presence of the FrontPage extensions indicates the possibility of this vulnerability. Please verify that your web server is running the necessary service packs to protect against this. Vulnerable systems: Default Installations of Windows NT4 IIS4 SP6 Default Installations of Windows 2000 IIS5 SP1 The vulnerability stems from Frontpage improperly handling queries to Frontpage Authoring (author.dll) modules as well as shtml calls. It is possible for a remote attacker to send a malformed query to those modules that will cause Frontpage to crash that will then in turn bring down inetinfo.exe on Windows NT 4.0 systems. On Windows 2000 systems, the vulnerability is a bit different. Inetinfo.exe is not killed, it just simply "freezes". You can still connect to the IIS5 web server but any further GET/HEAD/etc.. commands will not be processed. The two vulnerable Frontpage modules are: _vti_bin/shtml.dll/_vti_rpc/_vti_bin/_vti_aut/author.dll
Execution:	Exploiting this vulnerability involves sending several thousand characters in a POST request to the Frontpage server. Examples can be found here: http://www.eEye.com/html/advisories/FPDOSNT4.txt http://www.eEye.com/html/advisories/FPDOSNT4NT5.txt
Fix:	Patches are available from Microsoft: http://www.microsoft.com/technet/security/bulletin/ms00-100.asp
Reference:	eEye Advisory http://www.eeye.com/html/Research/Advisories/AD20001222.html Microsoft Advisory/Patch http://www.microsoft.com/technet/security/bulletin/ms00-100.asp Checkname: HTTP-IIS-FRONTPAGEAUTH

In addition, key information will be recorded in the Information part of the Site View window that can be added into the custom report using the Custom Report -> Application sections when generating the reports (see Figure 5-3).

5.2 ISS DATABASE SCANNER

This section is a supplement to the ISS Database User Guide and other ISS Database Scanner documentation. All material in this section is based on ISS Database Scanner version 4.2.

5.2.1 Tool Configuration

ISS Database Scanner is designed to run on a Windows platform while scanning Oracle, Sybase, and Microsoft SQL Server databases on a variety of UNIX and Windows-based systems. Upon starting the tool, there will be a “Welcome To Database Scanner” window with four options from which to choose: Scan Database, Set Security Policy, Review Results, and Password Strength.

Configure Connections

Before a scan of a database can be performed, the connections between ISS Database Scanner and the database to be tested must be configured. This effort is accomplished through the Configure Connections menu. To access this from the tool, select the Scanner -> Configure Connections menubar. Then, follow the instructions from the user guides in configuring a connection to the database type that is to be scanned.

Set SecurityPolicy

ISS Database Scanner performs database scans according to the security policy that is selected or developed. There are six built-in security policies, ranked as Levels Two through Seven. The Level One security policy acts as an inventory and classification of the overall system and is included as a separate policy within the tool.

The Level Two policy tests the risk of compromise from simplistic attacks. This policy checks for authentication vulnerabilities such as the following:

- Blank or easily guessed passwords
- Inappropriate registry security
- Default logins and passwords
- Inappropriate configuration settings
- Plaintext passwords in the registry or other files.

The Level Three policy tests for vulnerabilities to external system compromise from automated attack tools. This policy checks for authentication vulnerabilities such as the following:

- Brute force login attacks
- DoS attacks
- Protocol sniffing exploits.

The Level Four policy tests resistance to password cracking and susceptibility to external compromise from the very knowledgeable attacker. This policy checks for system integrity vulnerabilities such as the following:

- Buffer overflow exploits
- Authentication compromises in automated tools
- Compromises via Web tools or database access methods.

The Level Five policy tests resistance to local users gaining enhanced or system administrator privilege to access unauthorized or restricted information. This policy checks for system integrity vulnerabilities such as the following:

- Inappropriate ownerships
- Inappropriate permissions
- Inappropriate privileges
- Extended stored procedures and configuration settings that lead to unauthorized access.

The Level Six policy tests the integrity of the underlying operating system configuration on which the database is installed. This policy checks for system integrity vulnerabilities that could result in accidental or malicious changes, such as the following:

- Missing operating system service packs
- Missing operating system updates
- Missing operating system patches
- Unrecognized operating system file changes
- File permissions on critical operating system files.

The Level Seven policy tests the integrity of application data and customer specific application configurations against accidental or malicious changes. This policy checks for system integrity vulnerabilities such as the following:

- Missing operating system service packs, updates, or patches
- Missing database service packs, updates, or patches
- Poor audit trails
- Weak backup procedures
- Unrecognized database file changes
- Trojan horses in stored procedures.

A custom policy can also be developed. This can be accomplished by selecting the Set Security Policy button from the Welcome to Database Scanner window and then selecting “<New Security Policy>” from one of the three database policies and then the Open button. A new policy name should be typed in and the specific checks desired should also be checked.

Options

ISS Database Scanner options are accessible through the Scanner -> Options menubar. From the window that opens, there are four category tabs from which to choose: Licensing, Scanning Options, Sample Data, and Maintenance. The Licensing tab allows for licensing to be established for the databases that are licensed to for scanned. The Scanning Options tab allows for the configuration of the method used to query registry information along with the timeout for Database Scanner when waiting of results for the server being scanned. The query registry information method may be either *Use Extended Stored Procedures* or *Use Win32 API*. The timeout is in seconds and is valid only when scanning Microsoft SQL Server or Sybase databases. The Sample Data tab is for selecting either Sample Data or Live Data to use. Live Data should be used when performing an actual scan. Then, Maintenance tab allows for the compaction of the database the results are stored within. This can be implemented manually, or set up to run automatically based on the number of days since last compact, or when the database reaches a certain size.

5.2.2 Tool Use

ISS Database Scanner allows for three types of scanning:

- Full Audit Scan
- Penetration Testing
- Password Strength Test.

Full Audit Scan modes assess the database and operating system settings to determine whether the database server security complies with the specified security policy. This type of scan can also include the full range of Password Strength Tests. This type of scan requires the user to provide a password to access the selected database server.

Penetration Testing mode attempts to break into your database server without knowledge of a password. This method attempts to replicate the techniques that an attacker attempting to compromise the security of your database server might use. If the penetration testing mode is successful, there is the option to start a Full Audit Scan of the compromised database. (Note that the Penetration Test mode is accessed via the Scan Database menu.)

The **Password Strength Test** mode allows for password analysis checks to be run against the database without running a Full Audit Scan. These checks are the same ones within the Full Audit Scan, but run separately. The following password checks are run:

- Dictionary password attack
- No passwords on account
- Same password as account, with numbers appended at end
 - 1 to 100 for Oracle
 - 1 to 9 for Microsoft SQL Server
 - 1 for Sybase
- Password is reverse of account
- Same as server name (for Oracle).

To implement and run any of the above three types of scans, the developer should follow the instructions in the user guides to run a Full Audit Scan, Penetration Test, or Password Strength Tests. The Full Audit Scan and the Penetration Test can be accessed by selecting the *Scan Database* button, whereas the Password Strength Tests can be accessed by selecting the *Password Strength* button. When performing a Full Audit Scan, the settings shown in Table 5-6 are available.

Table 5-6. Full Audit Scan Options

Settings	Description
Server	Name of the server to scan
Trusted (SQL Server Only)	Use Windows NT authentication to make connection
Account	Account used to connect to the server for standard authentication
Password	Account Password used to connect to the server for standard authentication

Policy	Security policy used to scan the database (level 2 to 7, or custom)
Type	Type of Server (Microsoft SQL Server, Oracle, Sybase Adaptive Server)
Host Address	Host name or IP address of computer database server runs on
Host Account	Unix account or Windows NT user name for the operating system the database server runs on

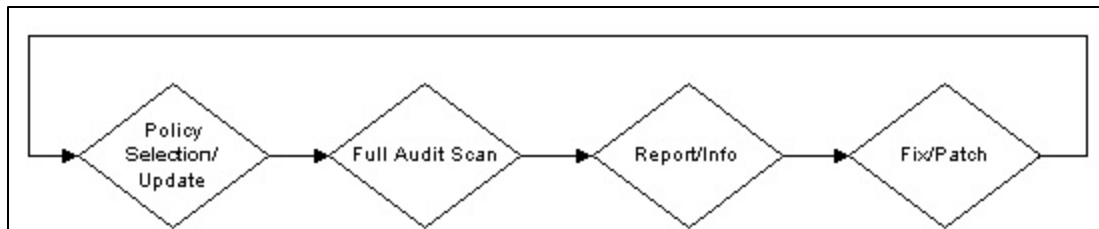


Figure 5-4. Database Scanner Usage

Host Password	Password for the designated Host User Name
---------------	--

To complete a full security test of a database, the Full Audit Scan mode should be used. In this mode, the tool will perform a full security audit of the database as set by the security policy. A custom security policy that has been developed for the database to be scanned should be used to ensure all required tests are performed. When performing a test, the following sequence of steps should be followed, looping back as necessary. (This sequence is illustrated in Figure 5-4.)

1. Policy Selection/Creation
2. Full Audit Scan
3. Report/Info
4. Fix/Patch.

If testing from an external standpoint is desired, the Penetration Testing mode should be used. In this mode, the tool will perform tests from an external standpoint with the option to complete a Full Audit Scan if internal access is found. The Full Audit Scan should use a custom security policy that has been developed for the database to be scanned should be used to ensure all required tests are performed. When performing a test, the following sequence of steps should be followed, as illustrated in Figure 5-5 to the right.

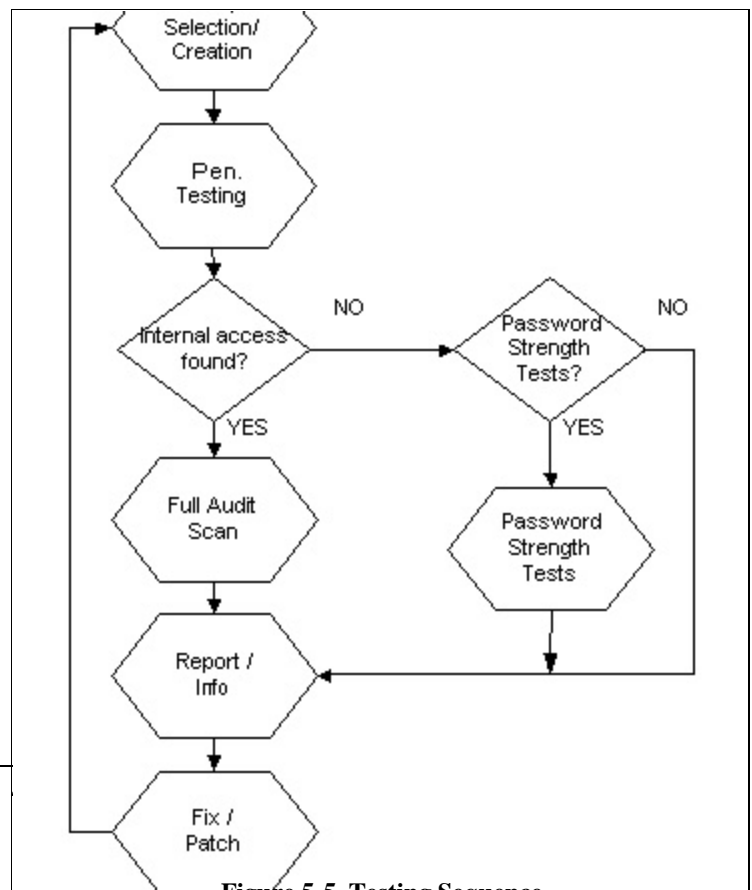


Figure 5-5. Testing Sequence

5.2.3 Interpreting Tool Output




To review the results, click the *Review Results* button from the main Welcome to Database Scanner window. Selecting from a list of possible types of reports or information to include in the overall report can then develop custom reports. The two main reports to view should be *Summary of Violations* and *Violations Details*. These reports will summarize the violations of the security policy that were identified in addition to providing detailed information for every violation identified. (See Figures 5-6 and 5-7.)


Violation Details

Server: ORACLE8

Policy: Maximum

Date of Scan: 4/6/00 2:18:57 PM

 - High Risk Violation
  - Medium Risk Violation
  - Low Risk Violation


Default Accounts and Passwords

Number of violations found for this check: 1

Detailed Description: Oracle databases have several well-known default username/password combinations. These combinations include the following: SCOTT/TIGER, DBSNMP/DBSNMP, SYSTEM/MANAGER, SYS/CHANGE_ON_INSTALL, TRACESVR/TRACE, CTXSYS/CTXSYS, MDSYS/MDSYS, DEMO/DEMO, CTXDEMO/CTXDEMO, APPLSYS/FND, PO8/PO8, NAMES/NAMES, SYSADM/SYSADM, ORDPLUGINS/ORDPLUGINS, OUTLN/OUTLN, ADAMS/WOOD, BLAKE/PAPER, JONES/STEEL, CLARK/CLOTH, AURORA\$ORBS\$UNAUTHENTICATED/INVALID, and APPS/APPS. These default combinations may provide unauthorized access to the server.

Fix: Change the default password to a value that is difficult to guess. Change the password for an account by executing the following command:
 ALTER USER <username> IDENTIFIED BY <new password>

Note: If you use Oracle Intelligent Agent and change the password for the DBSNMP account, you must also place the new password in the snmp_rw.ora file.

Violation Details:

The default password for the account DBSNMP has not been changed.

Figure 5-6. Violations Detail Example Report Example

A risk level of high, medium, or low is used to categorize the violations identified. Each violation identified will be listed in the *Violations Details* report with a detailed description of the violation along with fix information for correcting the violation.

After patching or repairing any discovered violations, a new scan should always be performed. This is especially important when certain violations are ruled as false-positives and are not patched. The results of this scan should subsequently be evaluated to ensure that the patching methodology used did not create any new violations. All new violations should be accepted or patched/repared. This process should continue until the application is brought to an acceptable risk level.

Additional reports can and should be generated. For more information about the different types of reports that can be generated, please see the documentation provided by ISS.

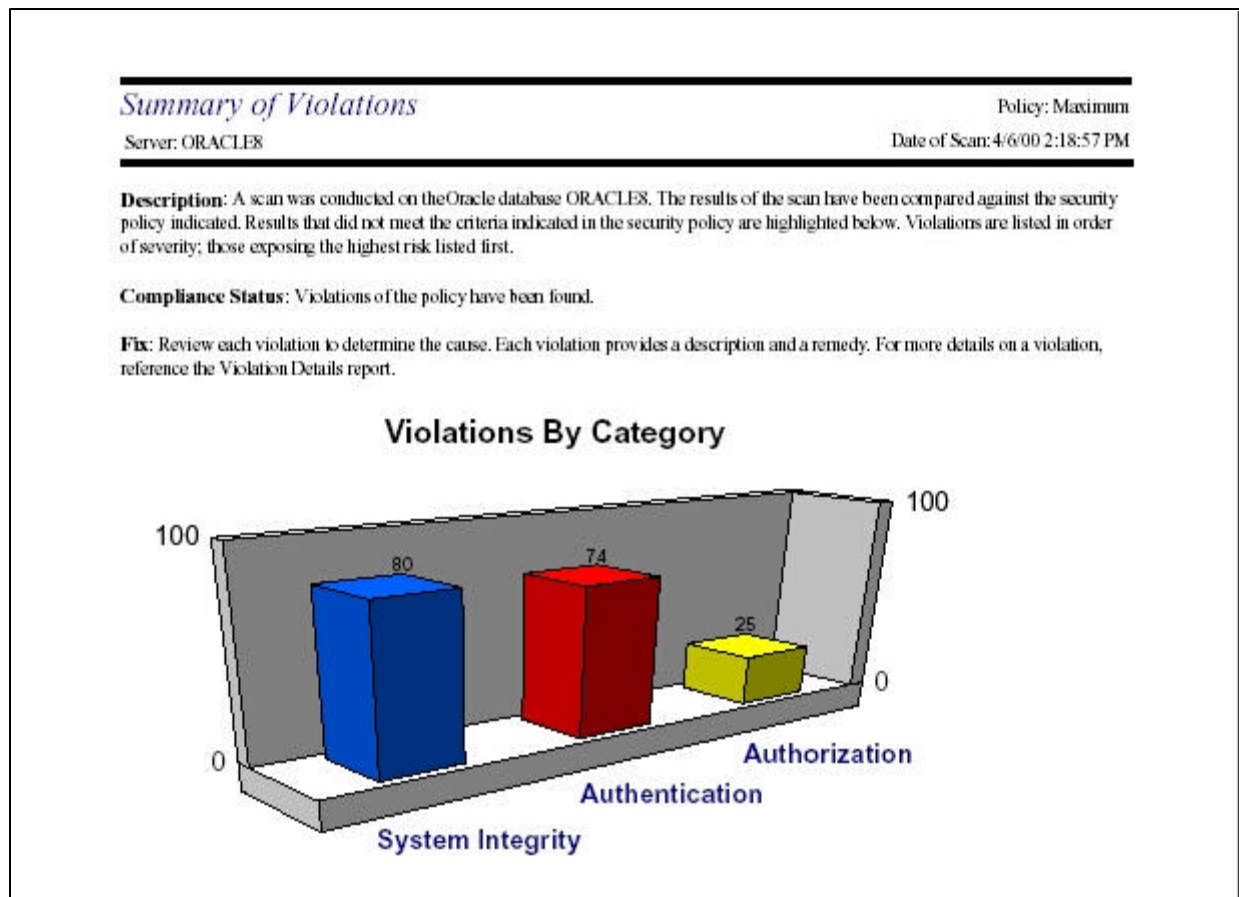


Figure 5-7. Summary of Violations Report Example

5.3 EEYE RETINA

This section is a supplement to the Retina Network Security Scanner document, which is also the user guide for the Retina scanner. All material in this section is based on Retina version 4.8.59.

5.3.1 Tool Configuration

This section further describes a normal configuration of the tool for use in a development environment (nonproduction). This tool is able to simulate active hacking techniques that are potentially dangerous. This tool should be used only in production environments if an expert user has customized and reviewed the tests to be run.

5.3.1.1 Update. Upon starting Retina, the user will be able to start an automatic update of the product, if this feature is enabled as described in the General Option in Section 5.3.1.5. These updates can be retrieved either through HTTP or HyperText Transfer Protocol Secure (HTTPS). It is recommended that the update be retrieved using HTTPS because this is a more secure method.

The user is not required to update, but updating is recommended because updates can increase the number of vulnerabilities that may be found and decrease the number of false positives that may be identified. If at any time during the use of the Retina an update is desired, this can be accomplished through the Tools->Updates menu bar. Note that this will require Retina to exit before the updates are retrieved; therefore, any current work should be stopped and saved completely before updates are attempted.

5.3.1.2 Target Selection. Retina target selection is accomplished with the use of IP addresses. Multiple IP addresses can be supplied to enable scanning of multiple hosts, if within license restrictions of the product. The IP addresses can be placed either within the address bar window, or using the menu bar option Edit->IP Range.

Note that when defining the address space via the address bar window, the addresses can be defined as those to be scanned and those to be excluded. All addresses placed in parentheses are those to be excluded, and the ones without are those to be scanned. In addition, URLs may be included in the address bar to be scanned.

For example, the following placed in the address bar window will scan all hosts in the range 192.168.0.1 to 192.168.0.32 except for 192.168.0.4 and 192.168.0.6 to 192.168.0.12. In addition, 192.168.0.64 will also be scanned.

Address: 192.168.0.1-192.168.0.32 (192.168.0.4) (192.168.0.6-192.168.0.12) 192.168.0.64

Custom scans can also easily be created using the Edit->IP Range feature. Using this feature, the following options shown in Table 5-7 become available to the user.

Table 5-7. Custom Scan Options

Command	Description
Add	Add the ranges entered to the address range to be scanned
Exclude	Exclude the ranges entered from the address range to be scanned
Delete	From the address list created, select any range and then Delete to remove
Clear	When creating a custom address range, this option will clear all addresses entered so far
Save	After a custom address range to be scanned is developed, it can be saved for later use
Load	If custom address ranges have been saved, this option allows them to be loaded

5.3.1.3 Policy Selection/Creation. The policies that define Retina's scans and tests can be controlled using the Tools->Policies menu bar. From this option, three areas can be edited: Policies, Ports, and Audits.

The **Policies** section is further separated into three parts: Policy selected, Preferences, and CHAM. The policy selected includes two default policies. The first is a Port Scan policy for accomplishing a complete port mapping of the targets, whereas the second is a Complete Scan that includes the complete port mapping and a complete vulnerability audit. By selecting one of the above default policies and then the *Add* button, custom policies can be derived from the default policy that was selected. This is the recommended course of action because all options in the default port scans or audits may not be desired in all networks.

The **Preferences** section contains to options:

- Force Scan
- Enable Connect Scan Mode.

The **Force Scan** option will proceed with any scan desired, even if the host does not respond to an initial ping. Not all networks will be configured to respond to pings, causing live hosts to appear as though they are not there. Therefore, it is recommended that this option be selected so hosts are not skipped over during testing.

The **Enable Connect Scan Mode** changes the default scan mode from a SYN scan mode to a CONNECT scan mode. The default SYN scan is a fast and reliable method for use in analyzing a network; however, certain conditions may cause problems with SYN scan functionality. These problems in functionality can arise as a result of firewall and network latency. If testing is performed in a development environment as recommended, there should not be any firewall or network latency problems. If CONNECT scanning is desired, this method is a more reliable, but slower method of scanning. CONNECT scanning mode should be enabled only if the user is positive that the SYN scanning is failing.

The CHAM test features are available for FTP, POP3, SMTP, and HTTP. Selecting the corresponding boxes enables the CHAM test features for the selected protocol. CHAM test features will be discussed further in Section 5.3.1.6.

The **Ports** section contains all of the defined ports for which Retina will scan for during a port scan. This is not a complete listing of all the ports from 1 to 65536, so a selection box is available to perform a complete port scan from 1 to 65356. A complete port scan will take additional time, but it is a proven method to identify any open ports. This is helpful in identifying unknown ports that should not be available and potentially are a security risk.

The **Audits** section contains the categories of vulnerability checks to be performed along with the specific checks within each category. These should all be reviewed and selected custom for what

checks are desired and level of safety in scanning is desired. The categories of vulnerability checks are shown in Table 5-8.

Within each of the categories, the specific individual checks may also be included or excluded from testing. If a category does not apply to the targets being scanned, or if a complete category is determined to be too risky to apply, the complete category can be removed from the policy at hand.

Table 5-8. Vulnerability Check Categories

Vulnerability Category	Description
Accounts	Includes all audits related to user accounts
CGI Scripts	Includes all audits of vulnerabilities associated with CGIs and scripts
CHAM	Includes all audits of vulnerabilities associated with CHAM
Commerce	Includes all audits of vulnerabilities associated with Commerce and store transaction servers
DNS Services	Includes all audits of vulnerabilities associated with DNS services
DoS	Includes all audits of vulnerabilities associated with DoS
FTP Servers	Includes all audits of vulnerabilities associated with FTP servers and file transfer protocols
IP Services	Includes all audits of vulnerabilities associated with IP services
Mail Servers	Includes all audits of vulnerabilities associated with Mail servers, POP3, Internet Message Access Protocol (IMAP), and SMTP
Miscellaneous	Includes all audits of vulnerabilities associated with miscellaneous vulnerabilities
NetBIOS	Includes all audits of vulnerabilities associated with the NetBIOS protocol
Registry	Includes all audits of vulnerabilities associated with the Windows Registry
Remote Access	Includes all audits of vulnerabilities associated to remote access software
RPC Services	Includes all audits relating to Remote Procedure Call (RPC) Services
Service Control	Includes all audits relating to software packages they may allow way to control and monitor their service remotely
SNMP Servers	Includes all audits of Simple Network Management Protocol (SNMP) server
SSH Servers	Includes all audits of Secure Shell (SSH) server
Web Servers	Includes all audits of vulnerabilities associated with Web servers, CGIs, and the HTTP

5.3.1.4 Ports. The Tools->Ports menu bar allows for the addition, deletion and editing of the standard ports that are included in the port scanning. This dialog allows for the custom definition of any port for TCP and UDP with a description of what protocol is defined for that port. The ports will be

listed by the port number, description, and then a letter denoting whether the description should be for TCP (T), UDP (U), or both (B).

5.3.1.5 Options. The options for the tool can be accessed from the Tools->Options menu bar. These settings include the following:

- General
- Alerts
- Schedule
- Theme.

The General options include settings for Performance, Reliability, Preferences, and Auto Update.

Performance

The Performance option allows for the performance level in relation to the speed and modules. The speed specifies the number of processes the modules will have available for use. This setting ranges from 1 to 20, with the speed of the scan increasing as the number increases. This means that a selection of 20 will provide the quickest scan. Note, however, that the higher numbers will require more resources, so a user should ensure that the host computer has sufficient resources available.

The modules allows for the control of how many targets can be scanned at the same time. The number selected corresponds to how many are allowed. As with speed, the higher the number, the faster the scan will complete; however, the same caveat exists in ensuring enough resources are available.

Reliability

The Reliability option allows for the ping timeout and data timeout to be specified. The ping timeout refers to the amount of time Retina will wait for a response from a machine it is trying to contact. This can range from 1 to 20, with a default of 10. During the initial ping scans that takes place to verify a computer is live, this option can be over ridden by the Force Scan option covered in Section 5.3.1.3. The data timeout refers to the amount of time Retina will wait for a response if a data transfer has stopped. This can range from 3 to 60, with a default of 30.

Preferences

The Preferences option allows a log to be created from the current scan operation. Additionally, it allows the tool to be minimized to the system tray as an icon. The log can be created by checking the *Create Log* box and the *Minimize to Tray* icon can be selected by checking the *Show as tray icon when minimized* box.

Auto Update

The Auto Update options allow for the auto update to never occur, to occur only after asking the user and to occur automatically. Because eEye could possibly make changes to the tool and what will be scanned, it is recommended that the update occur only after asking the user. Once the update occurs, changes to the tool and the vulnerabilities it detects should be reviewed before any new scans are started. Additional information about the Auto Update feature can be found in Section 5.3.1.1.

Alerts

The Alerts features allow for the configuration of local alerts, email alerts, and messenger alerts. This allows alerts to be sent through the above methods when vulnerabilities of the specified severity are identified by Retina. The severity specified is ranked as follows: info, low, medium, or high. Local alerts also allow for the playing of a sound file to audibly alert the user.

Schedules

The Schedule option allows for the scheduling of automatic scans for the local machine. By selecting days and times in a grid, scans can be scheduled as often as every hour if desired. Note that when scheduling automatic scans, the alert features should also be configured in a manner that will notify the user when vulnerabilities have been found so immediate action can be taken where necessary.

Themes

The Theme option allows for the modification of the default color theme of the tool. This option has no affect on the operation of the tool, only the preference of the user.

5.3.1.6 Common Hacking Attack Methods (CHAM). The CHAM feature of Retina is a collection of special audits referred to as the Common Hacking Attack Method audits. These audits are designed to discover unknown vulnerabilities within FTP, POP3, SMTP, and HTTP. The audits are designed to uncover potential problem areas that may be vulnerabilities as a result of configuration problems, buffer overflows, format bugs, integer overflows, off-by-one bugs, memory leaks, race conditions, signal handling bugs, memory allocation (malloc) corruption bugs, and a wide variety of other security holes that are exploited currently.

When testing in a development environment, these types of audits should always be included in the testing. Because this may discover unknown vulnerabilities, there may not always be readily available patching solutions. The solution may involve the contacting of the vendor responsible for the bug to resolve any potential risk identified.

5.3.1.7 Wizards

Audits/RTH Wizard

The Retina Audit Wizard is available from the Tools->Audit Wizards menu bar. This wizard allows the user to create a custom Retina audit, also known as an RTH. RTHs are files that contain information about a vulnerability that enables the Retina Scanner module to search for and identify it in vulnerable systems. Detailed information about creating these custom vulnerability checks can be found within the Retina User Guide.

Plugins Wizard

The Retina Plugin Wizard allows for modules written in the Retina API to be loaded or unloaded. Additional information about writing custom modules using the Retina API can be found in the Retina API documentation.

5.3.2 Tool Use

5.3.2.1 Tools Available. After configuring Retina and beginning the process to start the scanning, four main tools will be available to complete the process. The tools are as follows:

- Browser
- Miner
- Scanner
- Tracer.

The Browser tool allows the user to browse a Web address from the Retina interface. This tool can be separated into three parts: the Content view, the Details view, and the Outline view. The Content view is a browser within Retina that is an embedded version of Internet Explorer. The Details view contains information about the current Web page. This is the bottom window with the following information shown in Table 5-9.

Table 5-9. Details View Information

Information	Description
File Size	Size of the file displayed in Content view
File Created	Date file displayed in Content view was created
File Modified	Date file displayed in Content view was last modified
File Content	Text dump of the text on the file displayed in the Content view

The Miner tool is an agent released by Retina for the Retina AI Engine. This tool is an HTTP mining application that runs using the AI rules defined in a “Brain file” supplied by the Retina modules. The Brain file contains common words, file names, paths, and variables to be used by the Miner module in operations, such as password guessing and locating hidden or unknown Web pages. Any results found based on the Web server’s response to the Miner tool will be included in the report.

The Scanner Tool is Retina’s primary tool. This tool will perform the network audit of the addresses supplied based on the policy that has been selected. The scanner will search for all known open ports and services for the target IP addresses that are supplied. Based on the services found on the open ports, the scanner will test for the appropriate security vulnerabilities. All open ports, services found, and vulnerabilities identified can then be included in the report.

The Tracer tool completes a trace route from the host machine to the targets supplied, showing the results in a graphical format in the Content view window. This will also provide the user with all the IP address information of all the possible gateways, routers, and/or proxies between the host machine and the target machine that are identified in the network path.

5.3.2.2 Usage Description. This section will document the main usage of Retina as a scanner. In this mode, the tool will perform a network vulnerability scan of the IP addresses supplied to determine the risk levels of the hosts at each IP address. When performing a test, the following sequence of steps should be followed, looping back as necessary. (See Figure 5-8.)

1. Update
2. Scan
3. Report
4. Patch.

Update

Before starting any scan, it is important to always update the product. Because vulnerabilities are discovered daily, any updates to Retina's vulnerability database need to be downloaded and installed to ensure the latest checks are run when using Retina for a scan.

Scanner

The Scanner is Retina's default and primary mode for usage. When selected, the Scanner mode will have three main windows. These windows include the IP addresses that have been scanned, an overview of the findings for the IP address selected from the first window, and details for the finding selected in the second window. All the information found here will also be included in any reports generated by the tool.

To start a scan, select Tools->Scanner from the menu bar, select the address space to scan, and then press either the Enter key or the start button. This action will complete a scan based on the configuration settings and information entered. No further user interaction should be necessary to complete the scan.

Report/Info

Retina reports can be generated using the Tools->Reports button from the menu bar. From a popup dialog box, there are three main options that the user can select: Reports, Header Options, and Style Options. The Reports option allows for the selection of Complete Reports, Executive Reports, or Technical Reports. Note that custom reports can be developed by manually selecting and including each section of the report. In addition, each section of the report can be edited by selecting the Customize button.

The Header Options and Style Options sections allow for further customization of the reports. The Header Options allow for custom header logos, titles, and descriptions to be inserted into the reports. The Styles options allow for the style of report to be selected; however, the version of the tool at present has only one option available: the Modern Style.

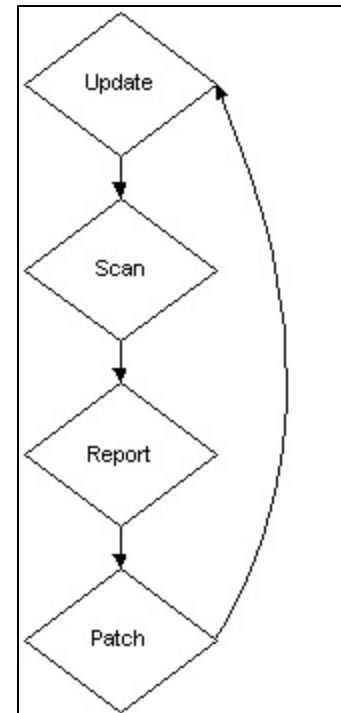


Figure 5-8. Retina Usage Flowchart

5.3.3 Interpreting Tool Output

A complete report generated by Retina will be discussed in this section. The reports generated by Retina include summary information along with detailed information on the findings separated out for each IP address that was scanned. (See Figure 5-9.)

The detailed information for each IP address includes the following:

- General
- Audits
- Machine
- Ports
- Services
- Shares
- Users

The **General** information includes such information as IP address, report date, domain name, ping response information, and time to live (TTL). This is basic information that does not go into any detail.

The **Audits** information includes the positive results of the security checks that were performed. Any time a high, medium, low, or information risk level vulnerability is identified, it will be listed in the Audits section.

The **Machine** information includes information such as the operating system, system time, MAC address, NetBIOS name, NetBIOS workgroup, closed ports, and open ports. This type of information is recorded in this area when found.

The **Ports** information includes any ports found to be listening or open and includes any suspected protocol (if known) being used. Note that this information is found during the scan of the ports identified when configuring the tool.

The **Services** information includes information such as computer browser, LanmanServer and LanmanWorkstation, Netlogon, RPC, and spooler information when available. The **Shares** information includes any shares that are identified. The **Users** information documents any user names that are identified during the scan process.

The most important information is contained in the Audits section for each IP address. This information details all vulnerabilities identified by Retina during the scanning process. This information is broken down into Risk Level, Description, How To Fix, and CVE sections. When available, reference URLs and BugtraqIDs are also included.

The Risk Level is broken down into four categories: high, medium, low, and information. Note that these risk levels are generically assigned by Retina and may not apply to all situations; therefore, the user must take care not to inadvertently dismiss them. All potential vulnerabilities must be evaluated to ensure

proper risk level assignment in relation to the application's environment and governing security policy. The other sections are self-explanatory and assist in identifying additional information pertinent to the vulnerability identified and possible remediation techniques and strategies.

After patching or repairing any discovered vulnerabilities, a new scan should always be performed. This is especially important when certain vulnerabilities are ruled as false-positives and are not patched. The results of this scan should subsequently be evaluated to ensure that the patching methodology used did not create any new vulnerabilities. All new vulnerabilities should be accepted or patched/repared. This process should continue until the application is brought to an acceptable risk level.

Audits: 192.168.005.025
CGI Scripts: TCP:80 - CGI - fpcount.exe Risk Level: High Description: A buffer overflow vulnerability in older versions of fpcount.exe, can be be remotely exploited to execute arbitrary commands. How To Fix: Upgrade the most recent version of Frontpage to confirm that the vulnerability has been eliminated. URL1: Microsoft Frontpage (http://microsoft.com/frontpage/) CVE: CAN-1999-1376
Web Servers: TCP:80 - IIS 5.0 IPP ISAPI Host overflow Risk Level: High Description: Due to an unchecked buffer in msw3prt.dll, a maliciously crafted HTTP .printer request containing approx 420 bytes in the 'Host' field will allow the execution of arbitrary code on unpatched Windows 2000 IIS 5.0 web servers. How To Fix: A patch is available from Microsoft to fix this vulnerability. We also recommend removing the .printer ISAPI filter if it is not needed. URL1: Microsoft - IPP Hotfix (http://www.microsoft.com/Downloads/Release.asp?ReleaseID=29321) URL2: eEye Digital Security Advisory (http://www.eeye.com/html/Research/Advisories/AD20010501.html) CVE: CVE-2001-0241 BugtraqID: 2674
Web Servers: TCP:80 - IIS Cumulative - ASP Chunked Encoding Overflow Risk Level: High Description: There exists a vulnerability within Microsoft IIS that allows an attacker to exploit a buffer overflow vulnerability stemming from improper handling of chunked encoding requests. How To Fix: Install the Microsoft patch. URL1: Microsoft Advisory (http://www.microsoft.com/technet/treeview/default.asp?url=/technet/security/bulletin/ms02-018.asp) URL2: eEye Digital Security Advisory (http://www.eeye.com/html/Research/Advisories/AD20020410.html) CVE: CAN-2002-0079

Figure 5-9. Retina Report Excerpt Example

5.4 SPLINT

This section is a supplement to the SPLINT Manual. The SPLINT Manual is well documented and a complete guide for using SPLINT. All material in this section is based on SPLINT version 3.0.1.6.

5.4.1 Tool Configuration

SPLINT is a tool for statically checking C programs for potential security and coding flaws. SPLINT (either Secure Programming Lint or Specifications Lint) which is based off LCLint, is developed and maintained by the Secure Programming Group at the University of Virginia Department of Computer Science.

SPLINT is executed from the command line and has many advanced features through the use of flags. These flags are well documented in Appendix B of the SPLINT Manual and will not be addressed here. Additional help is available for the topic areas shown in Table 5-10. If additional help is desired while using the tool, this can be accessed using the following command from the command line:

```
splint -help <topic or flag name>
```

Table 5-10. Topic Options with Help Available

Topic	Description
Annotations	Describes source-code annotations
Comments	Describes control comments
Flags	Describes flag categories
flags <category>	Describes flags in category
flags all	Short description of all flags
flags alpha	Lists all flags alphabetically
flags full	Full description of all flags
Mail	Information on mailing lists
Modes	Show mode settings
Parseerrors	Help on handling parser errors
Prefixcodes	Character codes in namespace prefixes
References	Sources for more information
Vars	Environment variables
Version	Information on compilation, maintainer

An example command of using SPLINT to test the c code contained in the file *test.c* without also testing any include files (with the extension *.h*) is as follows:

```
splint +never-include test.c
```

5.4.2 Tool Use

All scans performed should be done on copies of the source code in a secure directory. It should be ensured that the code scanned has not undergone any changes after scanning. If any changes are detected, then a new scan should be performed.

The tool may be used in its default state and used to scan code in an iterative process. Once the code is scanned, the results file should be analyzed. All potential security problems and issues should be reviewed and changed if possible. Once this action has been accomplished, another iteration of scanning should commence for two reasons: (1) the user needs to ensure that all security issues have been sufficiently patched, and (2) in the process of fixing one security problem, additional and new security issues may have been introduced into the code. Consequently, the scanning and fixing process should proceed iteratively until all security issues are resolved either through repair or dismissal by false positive.

5.4.3 Interpreting Tool Output

An example test of the c code *test.c* can be accomplished through the following process:

```
splint +never-include test.c >test-out.txt
```

(Note that this command will test the c code *test.c* and output the results in text format to the file *test-out.txt*. The resulting text file is Figure 5-10 below.)

```
Splint 3.0.1.6 --- 11 Feb 2002

test.c: (in function main)
test.c(10,2): Path with no return in function declared to return int
  There is a path through a function declared to return a value on which there
  is no return statement. This means the execution may fall through without
  returning a meaningful result to the caller. (Use -noret to inhibit warning)
test.c: (in function demo)
test.c(16,12): Unrecognized identifier: gettext
  Identifier used in code has not been declared. (Use -unrecog to inhibit
  warning)
test.c(17,9): Parameter 1 (b) to function strcpy is declared unique but may be
  aliased externally by parameter 2 (a)
  A unique or only parameter may be aliased by some other parameter or visible
  global. (Use -mayaliasunique to inhibit warning)
test.c(18,10): Unrecognized identifier: s
test.c(20,25): Unrecognized identifier: bug
test.c(21,2): Format string parameter to sprintf is not a compile-time
  constant: gettext("hello %s")
  Format parameter is not known at compile-time. This can lead to security
  vulnerabilities because the arguments cannot be type checked. (Use
  -formatconst to inhibit warning)
test.c(22,13): Unrecognized identifier: unknown
test.c(22,2): Format string parameter to sprintf is not a compile-time
  constant: unknown
test.c(23,9): Unrecognized identifier: bf
test.c(23,13): Unrecognized identifier: x
test.c(23,2): Format string parameter to printf is not a compile-time
  constant:
    bf
test.c(24,2): Return value (type int) ignored: scanf("%d", &x)
  Result returned by function call is not used. If this is intended, can cast
```



```

    result to (void) to eliminate message. (Use -retvalint to inhibit warning)
test.c(25,2): Return value (type int) ignored: scanf("%s", s)
test.c(26,2): Return value (type int) ignored: scanf("%10s", s)
test.c(27,2): Return value (type int) ignored: scanf("%s", s)
test.c(28,2): Use of gets leads to a buffer overflow vulnerability.  Use fgets
               instead: gets
    Use of function that may lead to buffer overflow. (Use -bufferoverflowhigh
to
    inhibit warning)
test.c(28,7): Unrecognized identifier: f
test.c(28,2): Return value (type char *) ignored: gets(f)
    Result returned by function call is not used. If this is intended, can cast
    result to (void) to eliminate message. (Use -retvalother to inhibit warning)
test.c(31,2): Use of gets leads to a buffer overflow vulnerability.  Use fgets
               instead: gets
test.c(31,2): Return value (type char *) ignored: gets(f)
test.c(32,2): Use of gets leads to a buffer overflow vulnerability.  Use fgets
               instead: gets
test.c(32,2): Return value (type char *) ignored: gets(f)
test.c(40,2): Path with no return in function declared to return int
test.c: (in function demo2)
test.c(49,3): Unrecognized identifier: _mbscopy
test.c(50,3): Function memcpy called with 2 args, expects 3
    Types are incompatible. (Use -type to inhibit warning)
test.c(50,12): Passed storage s not completely defined (*s is undefined):
               memcpy (... , s)
    Storage derivable from a parameter, return value or global is not defined.
    Use /*@out@*/ to denote passed or returned storage which need not be
defined.
    (Use -compdef to inhibit warning)
test.c(51,3): Unrecognized identifier: CopyMemory
test.c(52,3): Unrecognized identifier: lstrcat
test.c(53,3): Function strncpy called with 2 args, expects 3
test.c(54,3): Unrecognized identifier: _tcsncpy
test.c(57,3): Unrecognized identifier: _tcsncat
test.c(58,3): Assignment of size_t to int: n = strlen(d)
    To allow arbitrary integral types to match any integral type, use
    +matchanyintegral.
test.c(60,3): Unrecognized identifier: MultiByteToWideChar
test.c(60,23): Unrecognized identifier: CP_ACP
test.c(60,32): Unrecognized identifier: szName
test.c(60,42): Unrecognized identifier: wszUserName
test.c(73,3): Unrecognized identifier: SetSecurityDescriptorDacl
test.c(73,30): Unrecognized identifier: sd
test.c(75,3): Unrecognized identifier: CreateProcess
test.c(76,2): Path with no return in function declared to return int
test.c: (in function getopt_example)
test.c(81,13): Unrecognized identifier: optc
test.c(81,20): Unrecognized identifier: getopt_long
test.c(81,49): Unrecognized identifier: longopts
test.c(83,2): Path with no return in function declared to return int
test.c: (in function testfile)
test.c(88,10): Possibly null storage f passed as non-null param: fclose (f)

```

```

A possibly null pointer is passed as a parameter corresponding to a formal
parameter with no /*@null@*/ annotation.  If NULL may be used for this
parameter, add a /*@null@*/ annotation to the function parameter
declaration.
(Use -nullpass to inhibit warning)
test.c(87,7): Storage f may become null
test.c(88,3): Return value (type int) ignored: fclose(f)
test.c(89,2): Path with no return in function declared to return int
Finished checking --- 47 code warnings

```

Figure 5-10. SPLINT Output File Example

For additional information about interpreting the tool output and the various problems areas that are detected by SPLINT, see the manual that provides complete explanations.

5.5 FLAWFINDER

This section as a supplement to Flawfinder manual (man) page, or user guide. All material in this section is based on Flawfinder version 1.21.

5.5.1 Tool Configuration

Because Flawfinder is a script written in the Python programming language, installation of Python 1.5 or greater is required for the successful execution of Flawfinder. Flawfinder is designed find potential security flaws within C and C++ source code. Little configuration is needed for the successful operation of the tool.

In Flawfinder's most basic operation, the user must simply specify, at the command line, the directory of source code to scan. However, there are more advanced options implemented with the use of command line flags, as shown in Table 5-11.

Table 5-11. Flawfinder Command Line Flags

Flag	Description
--help	Show help information
--version	Show version of Flawfinder
--allowlink	Allow usage of symbolic links
--inputs	Show only functions that obtain data from outside the program
--minlevel=X	Set minimum risk level to X for inclusion in hitlist (0="no risk", to 5="maximum risk," with default of 1)
-m X	Same as --minlevel=X
--neverignore	Do not ignore lines when ignore comments are found
-n	Same as --neverignore

--columns	Show column number of each hit
--context	Show the context, or the text of line with potential hit
-c	Same as --context
--dataonly	Do not display header and footer
--html	Format output as HTML instead of text
--immediate	Immediately display hits, do not wait until end
-i	Same as --immediate
--singleline	Display as single line output for text for each hit
-S	Same as --singleline
--omittime	Omit timing information
--quiet	Do not display status information while analysis is on going
--loadhitlist= <i>F</i>	Load the hitlist from <i>F</i> instead of analyzing source file
--savehitlist= <i>F</i>	Save all resulting hits to filename <i>F</i>
--diffhitlist= <i>F</i>	Show only hits that are not in filename <i>F</i>

An example of the usage for the python script to test the c code *test.c* outputting only those vulnerabilities with a risk level of 2 or higher is as follows:

```
python flawfinder --minlevel=2 test.c
```

In addition to the flags, specially formatted comments may be added to source code so that certain lines of code are ignored during a Flawfinder audit. When these comments are present in the code, they can later be ignored using the *neverignore* flags as mentioned above. These comments can either be added on the same line, or placed by themselves on the previous line. The two types of formats in which the comments may be are as follows:

```
// Flawfinder.ignore
```

```
/* Flawfinder.ignore */
```

5.5.2 Tool Use

All scans conducted should be performed on copies of the source code in a secure directory to prevent any inadvertent corruption. It should be ensured that the code scanned has not undergone any changes after scanning. If any changes are detected, then a new scan should be performed.

The tool may be used in its default state and used to scan code in an iterative process. Once the code is scanned, the results file should be analyzed. All potential security problems and issues should be reviewed and changed if possible. Once this action has been accomplished, another iteration of scanning should commence. This is necessary for two reasons: (1) the user needs to ensure that all security issues have been sufficiently patched, and (2) in the process of fixing one security problem, additional and new security issues may have been introduced into the code. Consequently, the scanning

and fixing process should proceed iteratively until all security issues are resolved either through repair or dismissal by false positive.

5.5.3 Interpreting Tool Output

The tool output rates security issues on a scale of 0 to 5. By default, only those of level 1 and above will be reported. The risk level ranges from the level of 0, or no risk, up to the level of 5, or maximum risk. The tool output will be sorted by risk, with the highest risk findings listed first. All attempts should be made to reduce the risk level of all findings to 0, either through patching, or by manual elimination by careful research and false positive reduction.

Not all findings will be security vulnerabilities and simultaneously not all security vulnerabilities will be identified. This tool will assist only in identifying possibly security vulnerabilities, but as all these types of tools will be, it is not guaranteed to identify all possible security vulnerabilities.

A sample test of the c code contained in the file *test.c* can be accomplished through the following process and produce results as shown later.

```
python flawfinder --html test.c >test-out.html
```

This command will test the c code *test.c* and output the results in html format to the file *test-out.html*. The resulting html file is shown below in Figure 5-11:

Flawfinder Results

Here are the security scan results from [Flawfinder version 1.20](#), (C) 2001-2002 [David A. Wheeler](#). Number of dangerous functions in C/C++ ruleset: 127

Examining test.c

- test.c:32 [5] (buffer) *gets: does not check for buffer overflows. Use fgets() instead.*
- test.c:56 [5] (buffer) *strncat: easily used incorrectly; doesn't always \0-terminate or check for invalid pointers. Risk is high; the length parameter appears to be a constant, instead of computing the number of characters left.*
- test.c:57 [5] (buffer) *_tcsncat: easily used incorrectly; doesn't always \0-terminate or check for invalid pointers. Risk is high; the length parameter appears to be a constant, instead of computing the number of characters left.*
- test.c:60 [5] (buffer) *MultiByteToWideChar: Requires maximum length in CHARACTERS, not bytes. Risk is high, it appears that the size is given as bytes, but the function requires size as characters.*
- test.c:62 [5] (buffer) *MultiByteToWideChar: Requires maximum length in CHARACTERS, not bytes. Risk is high, it appears that the size is given as bytes, but the function requires size as characters.*
- test.c:73 [5] (misc) *SetSecurityDescriptorDacl: Never create NULL ACLs: an attacker can set it to*

Everyone (Deny All Access), which would even forbid administrator access.

- test.c:73 [5] (misc) *SetSecurityDescriptorDacl: Never create NULL ACLs; an attacker can set it to Everyone (Deny All Access), which would even forbid administrator access.*
- test.c:17 [4] (buffer) *strcpy: does not check for buffer overflows when copying to destination. Consider using strncpy or strlcpy (warning, strncpy is easily misused).*
- test.c:20 [4] (buffer) *sprintf: does not check for buffer overflows. Use snprintf or vsnprintf.*
- test.c:21 [4] (buffer) *sprintf: does not check for buffer overflows. Use snprintf or vsnprintf.*
- test.c:22 [4] (format) *sprintf: Potential format string problem. Make format string constant.*
- test.c:23 [4] (format) *printf: if format strings can be influenced by an attacker, they can be exploited. Use a constant for the format specification.*
- test.c:25 [4] (buffer) *scanf: the scanf() family's %s operation, without a limit specification, permits buffer overflows. Specify a limit to %s, or use a different input function.*
- test.c:27 [4] (buffer) *scanf: the scanf() family's %s operation, without a limit specification, permits buffer overflows. Specify a limit to %s, or use a different input function.*
- test.c:38 [4] (format) *syslog: if syslog's format strings can be influenced by an attacker, they can be exploited. Use a constant format string for syslog.*
- test.c:49 [4] (buffer) *_mbscopy: does not check for buffer overflows when copying to destination. Consider using a function version that stops copying at the end of the buffer.*
- test.c:52 [4] (buffer) *lstrcat: does not check for buffer overflows when concatenating to destination.*
- test.c:75 [3] (shell) *CreateProcess: this causes a new process to execute and is difficult to use safely. Specify the application path in the first argument, NOT as part of the second, or embedded spaces could allow an attacker to force a different program to run.*
- test.c:75 [3] (shell) *CreateProcess: this causes a new process to execute and is difficult to use safely. Specify the application path in the first argument, NOT as part of the second, or embedded spaces could allow an attacker to force a different program to run.*
- test.c:81 [3] (buffer) *getopt_long: some older implementations do not protect against internal buffer overflows. Check implementation on installation, or limit the size of all string inputs.*
- test.c:16 [2] (buffer) *strcpy: does not check for buffer overflows when copying to destination. Consider using strncpy or strlcpy (warning, strncpy is easily misused). Risk is low because the source is a constant string.*
- test.c:19 [2] (buffer) *sprintf: does not check for buffer overflows. Use snprintf or vsnprintf. Risk is low because the source has a constant maximum length.*
- test.c:45 [2] (buffer) *char: Statically-sized arrays can be overflowed. Perform bounds checking, use functions that limit length, or ensure that the size is larger than the maximum possible length.*
- test.c:51 [2] (buffer) *CopyMemory: does not check for buffer overflows when copying to destination. Make sure destination can always hold the source data.*
- test.c:87 [2] (misc) *fopen: Check when opening files - can an attacker redirect it (via symlinks), force the*

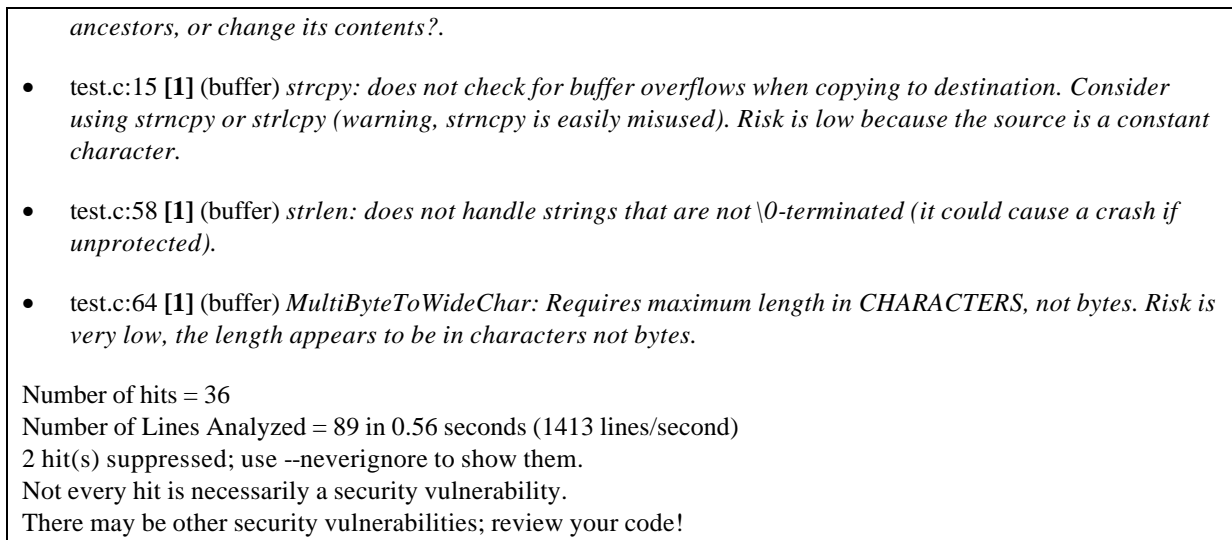


Figure 5-11. Example of Flawfinder Output File

As seen in Figure 5-11, all vulnerabilities identified are displayed from highest to lowest risk. The source code and line number at which the vulnerability was found are noted, followed by the risk level in brackets. The type of vulnerability is identified, followed by the vulnerability and a short description of the issue, sometimes included a method to correct the issue.

5.6 RATS

This section as a supplement to the RATS Readme page and other documentation. All material in this section is based on RATS version 2.0.

5.6.1 Tool Configuration

RATS is capable of scanning C, C++, Python, Perl, and PHP source code and identifying potential security vulnerabilities. This is accomplished by comparing function calls within the source code to potentially vulnerable functions calls within the databases supplied by RATS.

RATS is executed from the command line with configuration accomplished through the use of flags. The flags and their respective definitions are as shown in Table 5-12.

Table 5-12. RATS Command Line Flags

Flag	Description
-d <filename> --db <filename> --database <filename>	Specifies a vulnerability database to be loaded. Multiple databases may be loaded by repeating the use of the -d flag and filename
-h	Displays a brief usage help summary

--help	
-I --input	Lists function calls used that accepted external input at end of vulnerability report
-l <lang> --language <lang>	Force the specified language to be used regardless of filename extension. Valid language names for <lang> are c, perl, php, and python
-r --references	References to vulnerable function calls that are not being used as calls themselves are reported
-w <level> --warning <level>	Sets the warning level. Default is 2, valid selections are 1 (High), 2 (Medium), and 3 (Low).
-x	Causes the default vulnerability databases not to be loaded
-R --no-recursion	Disable recursion into subdirectories
--xml	Output in XML
--html	Output in HTML
--follow-symlinks	Evaluate and follow symlinks

An example of the usage for the RATS tool to test the c code contained in the file *test.c* which outputs only those vulnerabilities with a risk level of (3) or higher is as follows:

```
rats -warning 3 test.c
```

5.6.2 Tool Use

All scans performed should be performed on copies of the source code in a secure directory. It should be ensured that the code scanned has not undergone any changes after scanning. If any changes are detected, then a new scan should be performed.

The tool may be used in its default state and used to scan code in an iterative process. Once the code is scanned, the results file should be analyzed. All potential security problems and issues should be reviewed and changed if possible. Once this effort has been accomplished, another iteration of scanning should commence for two reasons: (1) the user needs to ensure that all security issues have been sufficiently patched, and (2) in the process of fixing one security problem, additional and new security issues may have been introduced into the code. Consequently, the scanning and fixing process should proceed iteratively until all security issues are resolved, either through repair or dismissal by false positive.

5.6.3 Interpreting Tool Output

The tool output rates the security risk of the findings as high, medium, and low risk. Within the output, this maps to a warning level of 1, 2, and 3, respectively. If a warning level of 1 is specified, then only default and high-risk level vulnerabilities will be included in the report. If either the default is kept, or a 2 is specified, then medium risk level vulnerabilities will also be included in the report, along with what is

included at the level of 1. If a warning level of 3 is specified, then the low risk level vulnerabilities will also be included in the report.

An example test of the c code *test.c* can be accomplished through the following process and produce results as shown later.

```
rats --warning 3 --html test.c >test-out.html
```

This command will test the c code contained in the file *test.c* and output the results in html format to the file *test-out.html*. The resulting html file is shown in Figure 5-12 below.

Note that not all findings will be security vulnerabilities and simultaneously not all security vulnerabilities will be identified. This tool will assist only in identifying possible security vulnerabilities, but is not guaranteed to identify all possible security vulnerabilities.

As shown in Figure 5-12, all vulnerabilities identified are output separately from highest to lowest risk. Each vulnerability is identified by the *Issue* field. This is followed by a short description of the issue at hand, and sometimes includes a method to correct the issue. The file tested and location of the vulnerability is then noted by the *File* and *Lines* fields.

Entries in perl database: **33**

Entries in python database: **62**

Entries in c database: **334**

Entries in php database: **55**

Analyzing **test.c**

RATS results

Severity: High

Issue: gettext

Environment variables are highly untrustable input. They may be of any length, and contain any data. Do not make any assumptions regarding content or length. If at all possible avoid using them, and if it is necessary, sanitize them and truncate them to a reasonable length. gettext() can utilize the LC_ALL or LC_MESSAGES environment variables.

File: **test.c**

Lines: 16 21

Severity: High

Issue: strcpy

Check to be sure that argument 2 passed to this function call will not copy more data than can be handled, resulting in a buffer overflow.

File: **test.c**

Lines: 16 17

Severity: High

Issue: sprintf

Check to be sure that the format string passed as argument 2 to this function call does not come from an untrusted source that could have added formatting characters that the code is not prepared to handle. Additionally, the format string could contain '%s' without precision that could result in a buffer overflow.

File: **test.c**
Lines: 20 21 22

Severity: High

Issue: sprintf

Check to be sure that the non-constant format string passed as argument 2 to this function call does not come from an untrusted source that could have added formatting characters that the code is not prepared to handle.

File: **test.c**
Lines: 21 22

Severity: High

Issue: printf

Check to be sure that the non-constant format string passed as argument 1 to this function call does not come from an untrusted source that could have added formatting characters that the code is not prepared to handle.

File: **test.c**
Lines: 23

Severity: High

Issue: scanf

Check to be sure that the format string passed as argument 2 to this function call does not come from an untrusted source that could have added formatting characters that the code is not prepared to handle. Additionally, the format string could contain '%s' without precision that could result in a buffer overflow.

File: **test.c**
Lines: 24 25 26 27

Severity: High

Issue: gets

Gets is unsafe!! No bounds checking is performed, buffer is easily overflowable by user. Use fgets(buf, size, stdin) instead.

File: **test.c**
Lines: 28 31 32

Severity: High

Issue: fixed size global buffer

Extra care should be taken to ensure that character arrays that are allocated on the stack are used safely. They are prime targets for buffer overflow attacks.

File: **test.c**
Lines: 45 46

Severity: High

Issue: _mbscopy

Check to be sure that argument 2 passed to this function call will not copy more data than can be handled, resulting in a buffer overflow.

File: **test.c**

Lines: 49

Severity: High

Issue: lstrcat

Check to be sure that argument 2 passed to this function call will not copy more data than can be handled, resulting in a buffer overflow.

File: **test.c**

Lines: 52

Severity: High

Issue: CreateProcess

Many program execution commands under Windows will search the path for a program if you do not explicitly specify a full path to the file. This can allow trojans to be executed instead. Also, be sure to specify a file extension, since otherwise multiple extensions will be tried by the operating system, providing another opportunity for trojans.

File: **test.c**

Lines: 75

Severity: Medium

Issue: SetSecurityDescriptorDacl

If the third argument, pDacl, is NULL there is no protection from attack. As an example, an attacker could set a Deny All to Everyone ACE on such an object.

File: **test.c**

Lines: 73

Severity: Low

Issue: memcpy

Double check that your buffer is as big as you specify. When using functions that accept a number n of bytes to copy, such as strncpy, be aware that if the dest buffer size = n it may not NULL-terminate the string.

File: **test.c**

Lines: 50

Severity: Low

Issue: CopyMemory

Double check that your buffer is as big as you specify. When using functions that accept a number n of bytes to copy, such as strncpy, be aware that if the dest buffer size = n it may not NULL-terminate the string.

File: **test.c**

Lines: 51

Severity: Low

Issue: strncpy

Double check that your buffer is as big as you specify. When using functions that accept a number n of bytes to copy, such as strncpy, be aware that if the dest buffer size = n it may not NULL-terminate the string. Also, consider using strlcpy() instead, if it is available to you.

File: **test.c**

Lines: 53

Inputs detected at the following points

Total lines analyzed: **90**

Total time **0.000000** seconds

0 lines per second

Figure 5-12. Example of RATS Output File

6. POST-ASSESSMENT PHASE

The post-assessment methodology is an essential component in the overall use and success of the Toolkit. The developer will not only need to correct and mitigate any vulnerabilities discovered, but also determine if the application still functions as originally intended (this test-evaluate-repair-validate cycle is illustrated in Figure 6-1). In addition, the developer must capture lessons learned and integrate security assessment and requirements validation into the overall life cycle of the application.

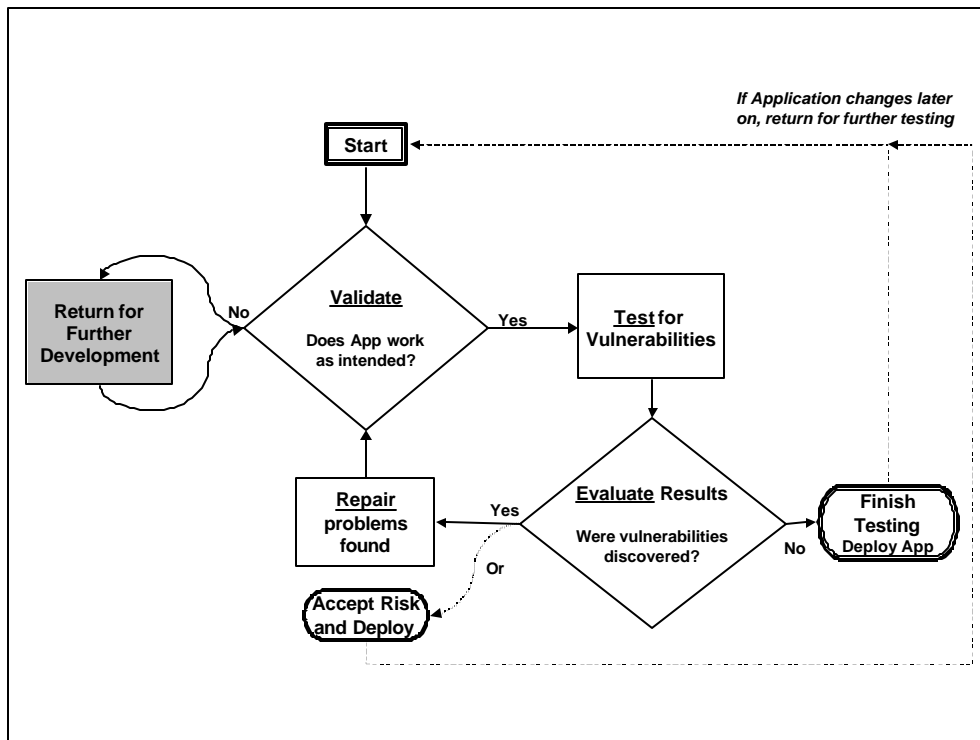


Figure 6-1. Test, Evaluate, Repair, Validate Cycle

6.1 TAKE CORRECTIVE ACTIONS AND APPLY SUGGESTED FIXES

Once a tool has successfully completed its scan of the application, a report documenting its results will typically be generated. If any vulnerabilities were discovered, it becomes the developer's responsibility to take appropriate corrective actions and implement any suggested fixes or repairs that were documented. (Note that most tools within the current Toolkit will not automatically correct or fix any vulnerabilities discovered. Instead corrective actions are suggested and documented to allow the developer to easily determine which actions to take and quickly implement.) Figure 6-2 illustrates of the type of output report that can be generated by many of the tools within the Toolkit. As shown in Figure 6-2, output reports typically summarize the vulnerability found and provide suggestions (e.g., URLs and citations) on where to locate other, more detailed information. In addition, the report might contain

suggestions on how to repair or mitigate the vulnerability. This information can range from URLs of missing software patches or the latest version of software, to suggestions on how to revise code to mitigate or eliminate the vulnerability.

Note that although the Toolkit will not specifically reference developer guidance or other related documentation, it behooves the developer to review this documentation for additional mitigation suggestions and strategies.

Vulnerabilities Report	
Server:	http://www.example.com:80
Severity: Critical	Since exploiting this vulnerability would involve ...
Count: 1	http://www.example.com/_vti_bin/_vti_aut/author.dll
Summary:	Since exploiting this vulnerability would involve crashing the web server, WebInspect does not perform false-positive checking for it. The presence of the FrontPage extensions indicates the possibility of this vulnerability. Please verify that your web server is running the necessary service packs to protect against this. Vulnerable systems: Default Installations of Windows NT4 IIS4 SP6 Default Installations of Windows 2000 IIS5 SP1 The vulnerability stems from Frontpage improperly handling queries to Frontpage Authoring (author.dll) modules as well as shtml calls. It is possible for a remote attacker to send a malformed query to those modules that will cause Frontpage to crash that will then in turn bring down inetinfo.exe on Windows NT 4.0 systems. On Windows 2000 systems, the vulnerability is a bit different. Inetinfo.exe is not killed, it just simply "freezes". You can still connect to the IIS5 web server but any further GET/HEAD/etc.. commands will not be processed. The two vulnerable Frontpage modules are: _vti_bin/shtml.dll/_vti_rpc/_vti_bin/_vti_aut/author.dll
Execution:	Exploiting this vulnerability involves sending several thousand characters in a POST request to the Frontpage server. Examples can be found here:
	http://www.eEye.com/html/advisories/FPDOSNT4.txt http://www.eEye.com/html/advisories/FPDOSNT4NT5.txt
Fix:	Patches are available from Microsoft: http://www.microsoft.com/technet/security/bulletin/ms00-100.asp
Reference:	eEye Advisory: http://www.eeye.com/html/Research/Advisories/AD20001222.html Microsoft Advisory/Patch: http://www.microsoft.com/technet/security/bulletin/ms00-100.asp Checkname: HTTP-IIS-FRONTPAGEAUTH

Figure 6-2. Sample Output Report

6.2 VALIDATE FUNCTIONALITY AND RERUN TOOL

Once actions have been taken to correct or mitigate any discovered vulnerabilities, it is vital that the developer retest the functionality of the application. This action would help assure the developer that none of the corrective actions have altered, damaged, or impaired the application's original functionality or have any unintended side effects. After completing this functionality test, the developer must go back and retest the application for vulnerabilities by re-running the assessment tool. If no vulnerabilities are discovered on this subsequent evaluation, then the developer may cease security testing. However, if vulnerabilities are discovered, then the developer must take further corrective actions, or accept the level of risk and cease security testing. If the developer chooses to take additional corrective actions, then the developer must subsequently retest the application for functionality and then retest again for vulnerabilities. This cycle of test-evaluate-repair-validate was illustrated in Figure 6-1.

6.3 CAPTURE LESSONS LEARNED

All vulnerabilities discovered, corrective actions taken, and lessons-learned should be captured by the development team as some form of a record, either written or electronic. This is important for two reasons. First, by capturing this information, the developer creates a library of knowledge about past vulnerabilities and corrective actions. This knowledge will help prevent similar vulnerabilities from reemerging. Second, this knowledge will help prevent new developers from repeating mistakes in the future, not only on the current application under development but also on any future applications. Any knowledge learned about secure programming practices and application development must be retained and passed onward to new developers.

The exact method of knowledge capture is left to the developer's discretion, but it is strongly recommended that it be performed in a clear, copious, and detailed manner that is easily accessible by all current and future developers on the team.

6.4 ESTABLISH A TESTING AND EVALUATION REGIME

Note that vulnerability assessment is not a finite and static event that is conducted once in the life cycle of an application. Instead, it is an ongoing process that must be repeatedly conducted throughout the entire life cycle of an application, from its early development to any incremental improvements made through time. Therefore, it is vitally important that developers establish a regular and routine testing regimen in which the application is tested for vulnerabilities and compliance with security requirements, especially after any development has occurred, major or minor. The use of such a testing and evaluation regime will help the developer assure that an application is secure.⁶

⁶ For more information on software development, engineering, and the software life cycle, visit Carnegie Mellon University's Software Engineering Institute (<http://www.sei.cmu.edu/sei-home.html>). For more information on security testing, see Chapter 2 of the *DRAFT Guideline on Network Security Testing*, John Wack and Miles Tracey, NIST Special Publication 800-42, Computer Security Division, National Institute of Standards and Technology, 2001. Although this document details security testing of networks, the same testing philosophy can be applied to application security testing.

7. CONCLUSION

As recent history has demonstrated, vulnerabilities are continually being discovered in all application categories. Only through a process of continual security assessment throughout the development life cycle can the developer have the slightest confidence that an application is secure. This document creates an assessment process for application developers and security engineers to use as they design and assess security mechanisms in their applications. This assessment methodology serves as a first step in ensuring that security is designed into applications. This methodology will help developers understand what needs to be secured so they can develop and insert specific application security controls and avoid creating vulnerabilities as applications progress through their development life cycle.

APPENDIX A: ACRONYMS

ACL	Access Control List
AES	Advanced Encryption Standard
AI	Artificial Intelligence
ANSI	American National Standards Institute
API	Application Program Interface
ASAT	Application Security Assessment Toolkit
CAC	Common Access Card
CGI	Common Gateway Interface
CHAM	Common Hacking Attack Methods
CIO	Chief Information Officer
CRL	Certificate Revocation Lists
CVE	Common Vulnerability Exposures
DBMS	Database Management System
DID	Defense in Depth
DISA	Defense Information Systems Agency
DNS	Domain Name Service
DoD	Department of Defense
DoS	Denial of Service
E-mail	Electronic Mail
FTP	File Transfer Protocol
GIG	Global Information Grid
GPL	GNU Public License
GUI	Graphical User Interface
HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
HTTPS	Hypertext Transfer Protocol Secure
I&A	Identification & Authentication
IIS	Internet Information Server
IMAP	Internet Message Access Protocol
IP	Internet Protocol
ISS	Internet Security Systems
KRL	Key Revocation Lists
LDAP	Lightweight Directory Access Protocol
MAC	Media Access Control
MAN	Manual
MS	Milliseconds
NetBIOS	Network Basic Input/Output System
NIST	National Institute Of Standards & Technology
NSA	National Security Agency
NTLM	NT LAN Manager for Microsoft Windows
OS	Operating System

PHP	PHP Hypertext Preprocessor
PKI	Public Key Infrastructure
PKE	Public Key Enabling
POP	Post Office Protocol
RATS	Rough Auditing Tool for Security
RBAC	Role Based Access Control
RPC	Remote Procedure Call
SHA	Secure Hash Algorithm
SMTP	Simple Mail Transfer Protocol
SNMP	Simple Network Management Protocol
SQL	Structured Query Language
SSH	Secure Shell
SSL	Secure Sockets Layer
SSO	Single Sign On
STIG	Security Technical Implementation Guide
TCP/IP	Transport Control Protocol / Internet Protocol
TOCTOU	Time Of Check/ Time Of Use
3DES	Triple Digital Encryption Standard
TTL	Time to Live
UDP	User Datagram Protocol
URL	Uniform Resource Locator
VA	Vulnerability Assessment
VAT	Vulnerability Assessment Tool
WinCGI	Windows Common Gateway Interface

APPENDIX B: REFERENCES

DRAFT Guideline on Network Security Testing, John Wack and Miles Tracy, NIST Special Publication 800-42, Computer Security Division, National Institute of Standards and Technology, 2001.

eEye Digital Security, makers of Retina, can be found at www.eeye.com

Flawfinder can be found at www.dwheeler.com/flawfinder/

Internet Security Systems, Inc., makers of Database Scanner, can be found at www.iss.net

RATS can be found at www.securesw.com/rats.php

Software Engineering Institute, Carnegie Mellon University, <http://www.sei.cmu.edu/sei-home.html>

SPI Dynamics, makers of WebInspect, can be found at www.spidynamics.com

Splint can be found at www.splint.org or <http://lclint.cs.virginia.edu>

APPENDIX C: REQUIREMENTS - TOOLKIT CAPABILITY MAP

The requirements in Appendix C were taken from the *Recommended Standard Application Security Requirements* document, Section 4. The following items are a quick description of each column in Table C-1.

- **Req#** is the reference number of the requirement. (This corresponds to the same requirement in the Recommended Standard Application Security Requirements guide).
- **QuickReference** is a simple phrase to quickly refer to the requirement.
- **Requirement** describes the actual requirement.
- **Vuln. Adr.** refers to the Common Application Vulnerabilities code number from the Recommended Standard Application Security Requirements guide, Section 3.1.1).
- **Tool Category Mapping** indicates all possible mappings of the requirement to the current toolkit.

Note that the right-most column, “Tool Category Mapping,” indicates all possible mappings for the current toolkit. If a tool category can be mapped to a requirement, then that mapping is explicitly listed in this column. Any category that is omitted for a particular requirement indicates that no mapping exists for that category at this time. For example, Requirement 4.0.1 indicates a direct mapping for the Web Application tool, Database tool, and General-Purpose tool, but **not** for the Developer tools. Consider Requirement 4.1.7, which indicates that an indirect mapping exists **only** for the Database tool; none of the other tools have a mapping (either direct or indirect). Finally, Requirement 4.0.7 has no mapping (direct or indirect) whatsoever; manual procedures must instead be used.

Table C-1. Requirements – Toolkit Capability Map

REQ #	QUICK REFERENCE	REQUIREMENT	VULN. ADR.	TOOL CATEGORY MAPPING
GENERAL APPLICATION SECURITY REQUIREMENTS FOR ALL IA MECHANISMS				
4.0.1	No bypass of application security controls	The application must prevent users from bypassing any security controls to directly access underlying Database Management System (DBMS) or operating system (OS) resources.	V3	<ul style="list-style-type: none"> • Web Application • Database • General-Purpose
4.0.2	Integrity of external security	The application must not perform, and must not be able to be used to perform, any function that may change the security configuration, security files, or security programs of the information system and operating environment to which the application belongs.	V2, V3	Indirect mapping using all tools.
4.0.3	Integrity of external operation	The application must not undermine or substitute the functionality or purpose of DBMS or OS files and programs, including security files and programs.	V2	Indirect mapping using all tools.
4.0.4	Integrity of platform security	The application must not modify underlying DBMS or OS security files, programs, or data.	V2	Indirect mapping using all tools.

REQ #	QUICK REFERENCE	REQUIREMENT	VULN. ADR.	TOOL CATEGORY MAPPING
4.0.5	Integrity of platform operation and data	The application must not modify files, programs, or data belonging to other applications, DBMS, or OS in any unauthorized way.	V2	Indirect mapping using all tools.
4.0.6	Interoperability with DoD PKI	The application's PKI functions must be implemented using PKI technology that is interoperable with DoD PKI. [7]	V3	None. Use Supplemental Procedures
4.0.7	Class 4 certificates	A PK-enabled application that supports DoD PKI Class 3 certificates must accommodate the transition to Class 4 certificates with minimal modification of application code. [8]		None. Use Supplemental Procedures
4.0.8	PK-enabling of applications	An electronic mail (E-mail) (including Web e-mail) application in a classified or unclassified environment, or a Web applications in an unclassified environment, must be PK-enabled by the deadline specified in DoD PKI Policy. [6,7] <i>NOTE: PK-enabling of Web applications in classified environments is strongly encouraged.</i>		None. Use Supplemental Procedures
4.0.9	Approval of cryptography	Encryption technology (including key lengths, algorithm, certificate class, and token type) used by the application in connection with I&A, access control, confidentiality, integrity, or nonrepudiation must be approved by National Security Agency (NSA) if the application is a Mission Category I application, or by NSA or National Institute of Standards and Technology (NIST) if the application is a Mission Category II application. [1,6,7].		None. Use Supplemental Procedures
4.0.10	High-risk services	Avoid use of high-risk services and technologies, such as Telnet, Simple Network Mail Protocol (SNMP) and mobile code in/by applications unless absolutely necessary.		<ul style="list-style-type: none"> • Web Application • General-Purpose
4.0.11	Application deployment	Before deploying the application, ensure that backup files, File Transfer Protocol (FTP) programs, and debugging files, tools, accounts, passwords, debug and test flags, and other developer "backdoors" have been removed from the application code and its platform.	V17	<ul style="list-style-type: none"> • Web Application • General-Purpose
4.0.12	Privileged processes	Avoid requiring application processes to have privileges greater than those granted to end-users of the application, unless absolutely necessary.		Indirect mapping using all tools.
4.0.13	Hypertext Markup Language (HTML) comments	Before installing the application, remove all references and comments from HTML code that might reveal features of the application's design.		<ul style="list-style-type: none"> • Web Application
4.0.14	Browser application facilities	Avoid use of scripts, cookies, and plug-ins unless absolutely necessary.		<ul style="list-style-type: none"> • Web Application
IDENTIFICATION AND AUTHENTICATION				
4.1.1	Authentication of application users	The application must ensure that users have been authenticated before granting them access to sensitive resources or trusted roles. [2,18].	V1	<ul style="list-style-type: none"> • Web Application • Database
4.1.2	Required Identification and Authentication	I&A performed before granting access to the application must use a nonforgeable, nonreplayable mechanism that supports one-way and two-way authentication. [2]	V1	None. Use Supplemental Procedures

REQ #	QUICK REFERENCE	REQUIREMENT	VULN. ADR.	TOOL CATEGORY MAPPING
	(I&A) technology			
4.1.3	Desirable I&A technology	I&A using single sign-on (SSO), public key enabling (PKE), smart card, and/or biometric technology is strongly encouraged. [1]	V1	None. Use Supplemental Procedures
4.1.4	Authentication chain of trust	For every user session and transaction, the application must ensure that an authentication chain of trust is established and maintained among the client/browser, the application server, and any backend servers used by the application.	V1, V3, V18	Indirect mapping using Web Application tool.
4.1.5	I&A trusted path	<i>If the application performs user I&A:</i> The I&A trusted path must be initiated by the user, not the application. [2]	V1	None. Use Supplemental Procedures
4.1.6	Backend system I&A	<i>If the server application interfaces with a “backend” system or DBMS, and that system requires users to be authenticated to it before allowing them access:</i> The application must not prevent the backend system from authenticating users as necessary.	V1, V3	None. Use Supplemental Procedures
4.1.7	Maximum number of unsuccessful I&A attempts	<i>If the application performs user I&A:</i> The I&A mechanism must enable administrator configuration of the maximum number of login attempts (configurable per user or per role) allowed within a given time period. [2]	V1	Indirect mapping using Database tool.
4.1.8	I&A lockout period	<i>If the application performs user I&A:</i> The application I&A mechanism must enable administrators to configure the duration of the “lockout” period during which a user (or role) who exceeds the number of allowable login attempts will be prevented from making another I&A attempt. [2]	V1	Indirect mapping using Database tool.
4.1.9	I&A using PKI certificates	<i>If the application performs certificate-based I&A:</i> The application must support the PKI (X.509) certificate class appropriate to the application’s Mission Category [1,2,7]: <ul style="list-style-type: none"> • <i>Mission Categories I and II:</i> DoD PKI Class 4 certificates, or Class 3 certificates (until the deadline for DoD PKI transition to Class 4) • <i>Mission Category III:</i> DoD PKI Class 3 (until DoD PKI transition to Class 4) 	V1	None. Use Supplemental Procedures
4.1.10	I&A using PKI tokens	<i>If the application performs token-based I&A:</i> The application must support the type of token appropriate to the application’s Mission Category [1,2,7]: <ul style="list-style-type: none"> • <i>Mission Categories I and II:</i> FORTEZZA, common access card (CAC), or another NSA-approved Class 3 or Class 4 hardware token • <i>Mission Category III:</i> CAC or software token (until DoD PKI transition to CAC) 	V1	None. Use Supplemental Procedures

REQ #	QUICK REFERENCE	REQUIREMENT	VULN. ADR.	TOOL CATEGORY MAPPING
4.1.11	Private Web server I&A	<i>If the application is a Web server application running on an Unclassified but private (versus public access) Web server:</i> The Web server application must perform I&A using Class 3 or Class 4 PKI (X.509) certificates transmitted via Secure Sockets Layer (SSL). [2,6,7]	V1	None. Use Supplemental Procedures
4.1.12	Public Web server I&A	<i>If the application is a Web server application running on a Web server that stores publicly releasable data to which access must be restricted to either:</i> <ul style="list-style-type: none"> • Preserve copyright protections, or • Limit access to only browsers originating from certain sites, or • Facilitate development of the data: The Web server application must implement I&A using Class 3 or Class 4 PKI (X.509) certificates transmitted via SSL. [2,6,7]	V1	None. Use Supplemental Procedures
4.1.13	Classified Web server I&A	<i>If the application is a Web server application running on a classified Web server:</i> The Web server application should implement certificate-based I&A using the appropriate class of PKI (X.509) certificate. [6,7]	V1	None. Use Supplemental Procedures
4.1.14	Browser support for tokens	Browsers, including those that support software tokens, must support use of CAC or FORTEZZA (as appropriate for the particular application) for storing the user's certificates, by the DoD PKI-defined deadline for migration to tokens. [6,7]	V1	None. Use Supplemental Procedures
4.1.15	No I&A by Java applets	Web applications must not use Java applets to perform I&A.	V1	None. Use Supplemental Procedures
4.1.16	Support for Class 4 certificates	<i>If the application performs I&A based on Class 3 certificates:</i> The application must be able to accommodate use of Class 4 certificates with minimal change to the application code. [1,2,7]	V1	None. Use Supplemental Procedures
4.1.17	Support for CAC	<i>If the application performs I&A based on certificates stored in software tokens:</i> The application must be able to accommodate use of the CAC by the DoD PKI policy-defined deadline, with minimal change to the application code. [1,7]	V1	None. Use Supplemental Procedures
4.1.18	I&A using biometrics	<i>If the application performs user I&A using biometrics:</i> The application I&A mechanism shall use biometrics in accordance with DoD policy. [1] <i>NOTE: As of May 2002, no DoD biometric policy has been published.</i>	V1	None. Use Supplemental Procedures
4.1.19	Strong passwords	<i>If the application performs user I&A based on UserID and static password:</i> The application's password management mechanism must prevent users from choosing passwords that do not comply with the following rules	V1, V4	None. Use Supplemental Procedures

REQ #	QUICK REFERENCE	REQUIREMENT	VULN. ADR.	TOOL CATEGORY MAPPING
		<p>[1,2]:</p> <ol style="list-style-type: none"> 1. Password must be case-sensitive 2. Password must contain at least eight characters 3. Password must not contain spaces or "+" 4. Password must contain at least one of each of the following: <ul style="list-style-type: none"> • Uppercase letter, and • Lowercase letter and • Nonalphanumeric ("special") character <p><i>NOTE: An example of a correctly constructed password:</i></p> <p>[etMe1n!]</p>		
4.1.20	Password changes	<p><i>If the application performs user I&A based on UserID and static password:</i></p> <p>The application's password management mechanism must [1,2]:</p> <ol style="list-style-type: none"> 1. Enable the administrator to assign passwords to users. 2. Enable the user to change his/her own password on demand. 	V1	None. Use Supplemental Procedures
4.1.21	Password expiration	<p><i>If the application performs user I&A based on UserID and static password, and the application does not handle only publicly releasable data (in which case, this requirement is optional):</i></p> <p>The application's password management mechanism must enable the administrator to set an expiration threshold for every password associated with every UserID. [1,2]</p>	V1	• Database
4.1.22	Selection of new password after password expiration	<p><i>If the application performs user I&A based on UserID and static password, and the application does not handle only publicly releasable data (in which case, this requirement is optional):</i></p> <p>The application must not authenticate a user whose password has expired until the user changes the expired password. [1,2]</p>	V1	• Database
4.1.23	Group I&A	<p><i>If the application performs I&A at the group level:</i></p> <p>The application must individually authenticate each user who claims to be a group member before performing group level I&A. [1]</p>	V1	Indirect mapping using Web Application and Database tools.
4.1.24	Confidentiality of transmitted passwords	<p><i>If the application transmits sensitive I&A data (e.g., passwords) over a network:</i></p> <p>The application must encrypt user passwords and any other sensitive I&A data before transmission over a network. [1,2,18]</p> <p><i>NOTE: Use of hexadecimal or another noncryptographic encoding scheme instead of encryption is unacceptable.</i></p>	V1, V5, V18	None. Use Supplemental Procedures
4.1.25	Confidentiality of password during reformatting	<p><i>If the application reformats passwords or other sensitive I&A data:</i></p> <p>The application must prevent any other process or user from reading the cleartext I&A data while they are being reformatted.</p>	V1, V5	None. Use Supplemental Procedures

REQ #	QUICK REFERENCE	REQUIREMENT	VULN. ADR.	TOOL CATEGORY MAPPING
4.1.26	Integrity of I&A data	<p><i>If the application reformats or transmits over a network passwords or other sensitive I&A data:</i></p> <p>The application must—</p> <ol style="list-style-type: none"> 1. Change only the presentation format of the I&A data, and not the data content. 2. Prevent any other process or user from modifying the I&A data during reformatting or transmission. 	V1, V6	<ul style="list-style-type: none"> • Web Application
4.1.27	I&A between client and server processes	<p><i>If the application is a distributed client-server application that:</i></p> <ul style="list-style-type: none"> • Operates in a high level of concern environment in which security protections are not adequately robust, and • Performs sensitive functions or handles high-value information. <p>The application's server processes must authenticate all client processes using an interprocess authentication technology approved by NSA or NIST and appropriate for the application's Mission Category (e.g., X.509/SSL or Kerberos), before accepting any requests from those client processes.</p>		None. Use Supplemental Procedures
4.1.28	Interprocess I&A in peer-to-peer applications	<p><i>If the application is a distributed peer-to-peer application that:</i></p> <ul style="list-style-type: none"> • Operates in a high level of concern environment in which security protections are not adequately robust, and • Performs sensitive functions or handles high-value information. <p>The application's peer processes must mutually authenticate one another using an interprocess authentication technology approved by NSA or NIST and appropriate for the application's Mission Category (e.g., X.509/SSL or Kerberos) before either process accepts any request from the other process.</p>		None. Use Supplemental Procedures
4.1.29	Warning message to authenticated users	<p><i>If the application performs user I&A:</i></p> <p>The application must notify every authenticated user of the following before granting the user access to the application resources [1,2]:</p> <ol style="list-style-type: none"> 1. The user has accessed a government system 2. The extent to which the application will protect the user's privacy rights 3. The highest sensitivity level/classification of data that may be handled by the application 4. The user's actions are subject to audit 5. The user's responsibilities for handling sensitive or classified information when using the application. <p>Furthermore, if the application handles classified information, the notification message must also include the following information associated with the authenticated UserID:</p> <ol style="list-style-type: none"> 6. Date, time, origination (e.g., client/browser IP address or 		None. Use Supplemental Procedures

REQ #	QUICK REFERENCE	REQUIREMENT	VULN. ADR.	TOOL CATEGORY MAPPING
		domain name) of most recent previous login 7. Number of unsuccessful login attempts by the UserID since the last successful login.		
4.1.30	Unique UserIDs	<i>If the application performs user I&A based on UserID and password:</i> The application must not accept more than one password from the same UserID.	V1	None. Use Supplemental Procedures
4.1.31	Unique passwords	<i>If the application performs user I&A based on UserID and password:</i> The application must not accept the same password from more than one UserID.	V1	None. Use Supplemental Procedures
4.1.32	No anonymous accounts	The application must not authenticate anonymous UserIDs.	V1	None. Use Supplemental Procedures
4.1.33	Authentication requires trustworthy credential	The application must not authenticate users based on UserID alone; the application must require users to present a trustworthy authentication credential (e.g., password, certificate, and biometric).	V1	None. Use Supplemental Procedures
4.1.34	Freedom in assigning UserIDs and GroupIDs	The application must not prevent the administrator from assigning any UserID he/she chooses to any user account, or from assigning any GroupID he/she chooses to any group account. The application must not force the administrator to assign a particular UserID to a particular user account (e.g., "Administrator" to the administrator account), and must not force the administrator to assign a particular GroupID to a particular group account.	V1	None. Use Supplemental Procedures
AUTHORIZATION AND ACCESS CONTROL				
4.2.1	Authorization	The application must ensure that users have been authorized to perform the functions they attempt to perform or access the resources they attempt to access, and that those authorizations explicitly allow them to perform those functions or access those resources in the ways the users attempt to do so.	V2	<ul style="list-style-type: none"> Web Application Database
4.2.2	Authorization information management	<i>If the application performs authorization:</i> The application must provide a tool for creating and modifying authorization information (e.g., Access Control Lists [ACLs]) and must ensure that the tool can be accessed only by an authorized user.		None. Use Supplemental Procedures
4.2.3	Authorization information confidentiality	<i>If the application performs authorization:</i> The application must protect the confidentiality of its authorization information. [19]	V2	Indirect mapping using all tools.
4.2.4	Authorization information integrity	<i>If the application performs authorization:</i> The application must protect the integrity of its authorization information from unauthorized modification or substitution.	V2	<ul style="list-style-type: none"> Web Application
4.2.5	Authorization information availability	<i>If the application performs authorization:</i> The application must protect the availability of its authorization information.	V2	Indirect mapping using all tools.

REQ #	QUICK REFERENCE	REQUIREMENT	VULN. ADR.	TOOL CATEGORY MAPPING
4.2.6	Interprocess authorization	<i>If the application performs interprocess I&A:</i> The application must perform interprocess authorization using technology (e.g., X.509 certificates) approved by NSA or NIST and appropriate for the application's Mission Category.		None. Use Supplemental Procedures
4.2.7	RBAC for privileged accounts	The application must ensure that Role-Based Access Control (RBAC) is used to designate and authorize privileged accounts (e.g., administrator accounts).	V2	None. Use Supplemental Procedures
4.2.8	RBAC in classified applications	Classified applications must ensure that RBAC enforces separation of duties and least privilege. [1]	V2	None. Use Supplemental Procedures
4.2.9	Maximum number of sessions	<i>If the application that allows a multiple sessions by the same UserID:</i> The application must [1,2]: <ol style="list-style-type: none"> 1. Enable the administrator to configure the maximum number of simultaneous sessions allowable per UserID, per role, and per organization/Group ID. 2. Prevent users who reach that maximum from initiating another session until they terminate an already-active session. 		None. Use Supplemental Procedures
4.2.10	Inactivity timeout ("deadman" capability)	The application must [1,2]: <ul style="list-style-type: none"> • Enforce a session timeout that suspends user access to the application after a configured period of inactivity. • Reauthenticate the user before allowing him/her to resume a suspended session. • Allow the administrator to configure the session timeout. • Allow the user to suspend his/her application session at will. 	V2	None. Use Supplemental Procedures
4.2.11	Access control for classified data	<i>If the application (regardless of Mission Category):</i> <ul style="list-style-type: none"> • Stores classified data, and • Can be accessed by users who are not cleared to read data of that classification level The application must [7]: <ul style="list-style-type: none"> • Provide the necessary APIs to an underlying OS (and, if appropriate, DBMS) that implements Mandatory Access Controls (MAC) robust enough to protect the classified data from unauthorized disclosure, or • Use NSA-approved Type 1 cryptography to encrypt the data before storage. 	V2	None. Use Supplemental Procedures
4.2.12	Access control for sensitive and Mission Category I unclassified data	<i>If the application:</i> <ul style="list-style-type: none"> • Stores sensitive* or Mission Category I unclassified data, and • Can be accessed by users who are not authorized to read 	V2	None. Use Supplemental Procedures

REQ #	QUICK REFERENCE	REQUIREMENT	VULN. ADR.	TOOL CATEGORY MAPPING
		<p><i>those data.</i></p> <p>The application must [7]:</p> <ul style="list-style-type: none"> • Provide the necessary Application Programming Interface (API)s to an underlying OS (and, if appropriate, DBMS) that implements access controls robust enough to protect the classified data from unauthorized disclosure; or • Use 3 Data Encryption Standard (3DES) or Advanced Encryption Standard (AES) to encrypt the data before storage. 		
4.2.13	Data change notification	The application must indicate to users, upon access to data, the date and time of the most recent change to the data. [1]		None. Use Supplemental Procedures
4.2.14	Labeling of classified data	<p><i>If the application is either:</i></p> <ul style="list-style-type: none"> • <i>Used to create or modify classified data, or</i> • <i>A Mission Category I is used to create or modify classified or sensitive data.</i> <p>The application must apply the appropriate confidentiality and integrity labels to the data at the time of creation or modification. These labels must be understood by the access control mechanism used to control access to the data. [1,2]</p>		None. Use Supplemental Procedures
4.2.15	Labeling of unclassified data	<p><i>If the application is used to create or modify unclassified but not publicly releasable data:</i></p> <p>The application must apply a label to the data upon creation or modification that clearly indicates that the data are not releasable to the public. These labels must be understood by the access control mechanism used to control access to the data. [1,2]</p>		None. Use Supplemental Procedures
4.2.16	Marking of output	<p><i>If the application transmits data or sends it to a printer:</i></p> <p>The application must ensure that the data are marked to reflect the sensitivity level/classification of data produced by the application (including handling caveats, code words, and dissemination control markings). [2]</p>		None. Use Supplemental Procedures
4.2.17	Invalid pathname references	Whenever a pathname or Uniform Resource Locator (URL) referenced in the application code is changed or removed from the system, the application code must be changed to change or delete that reference.	V14	• Web Application
4.2.18	Truncated pathnames	<p><i>If a user presents a truncated pathname or URLs that do not end in a file name:</i></p> <p>The application must not allow the user to access the file system directory indicated by the pathname.</p>	V13	• Web Application
4.2.19	Relative pathnames	<p><i>If the application code contains references to pathnames or URLs:</i></p> <p>The code must reference the absolute pathname/URL, not the relative pathname or URL.</p>	V12	• Web Application
4.2.20	Relative pathnames	Applications must not accept relative pathnames or URLs input by	V12	• Web Application

REQ #	QUICK REFERENCE	REQUIREMENT	VULN. ADR.	TOOL CATEGORY MAPPING
	input by users	users.		
4.2.21	Rejection of directly entered URLs	<i>If the application is a Web portal application:</i> The application must not allow users to access Web pages/resources not explicitly allowed by links on the portal page by directly typing the URLs of the forbidden pages/resources into their browser's "Location" line.	V20	• Web Application
4.2.22	Browser protection of user identity information	Browsers should ensure that cookies and other user identity information stored on the browser platform are protected from disclosure and tampering.	V19	• Web Application
4.2.23	CGI scripts	CGI scripts must not contain "holes" that can be exploited to gain direct access to the underlying operating system.	V16	• Web Application
CONFIDENTIALITY				
4.3.1	Encryption API	There must be an API that enables the application to invoke an encryption capability to selectively encrypt data and files. [2]		None. Use Supplemental Procedures
4.3.2	Nondisclosure of cleartext data	The application must ensure that sensitive cleartext data (including passwords) are not disclosed before encryption.	V5, V18	None. Use Supplemental Procedures
4.3.3	Encryption before transmission	<i>If the application transmits data over a network:</i> The application must invoke encryption of data before transmission using encryption technology appropriate for the characteristics of the data and the network, as indicated below [1,2,7]: 1. <i>If the data are classified at a higher level than the network.</i> Invoke NSA-approved Type 1 encryption technology. 2. <i>If there are users on the network who are not cleared to read data at that classification level:</i> Invoke NSA-approved Type 1 encryption technology. 3. <i>If the network is a public network:</i> Invoke NSA-approved Type 1 encryption technology. 4. <i>If the data are SAMI data (even if the network is at the same classification level as the data):</i> Invoke NSA-approved Type 1 encryption technology. 5. <i>If the network is at the same classification level as the data, BUT the data have a different need-to-know than the network:</i> Invoke 3DES or AES, or get a signed waiver from the data owner allowing unencrypted transmission. 6. <i>The data are sensitive, and the network is a public network.</i>	V5	None. Use Supplemental Procedures

REQ #	QUICK REFERENCE	REQUIREMENT	VULN. ADR.	TOOL CATEGORY MAPPING
		<p>Invoke 3DES or AES (shorter key length is acceptable)</p> <p>7. <i>The data are Mission Category I data, and the network is a public network.</i></p> <p>Invoke 3DES or AES with the longest key length.</p>		
4.3.4	Encryption of stored data	<p><i>If the infrastructure on which the application runs does not adequately protect the confidentiality of sensitive data stored by the application:</i></p> <p>The application must invoke encryption of data before storage using encryption technology appropriate for the characteristics of the data and the network, as indicated below:</p> <ol style="list-style-type: none"> <i>If the data are classified and the application can be accessed by users not cleared to data of that classification level:</i> <p>Invoke NSA-approved Type 1 encryption technology.</p> <ol style="list-style-type: none"> <i>If the data are Sensitive or Mission Category I Unclassified and the application can be accessed by users not authorized to read those data:</i> <p>Invoke 3DES or AES [7].</p> <ol style="list-style-type: none"> <i>If the data are SAMI, and the application can be accessed by users not cleared/authorized to read SAMI, and no waiver has been granted by the responsible Chief Information officer (CIO) to allow the data to be stored in unencrypted form:</i> <p>Invoke NSA-approved Type 1 encryption technology. [1]</p>		None. Use Supplemental Procedures
4.3.5	Protection of cryptokeys	<p><i>If the application ensures that data are encrypted:</i></p> <p>The encryption facility invoked by the application must ensure that unauthorized users cannot access the cryptokeys needed to decrypt the data. [2]</p>		None. Use Supplemental Procedures
4.3.6	PKI encryption certificates	<p><i>If the application invokes a PKI to encrypt data:</i></p> <p>The PKI invoked by the application must use DoD PKI Class 4 or Class 3 certificates when performing the encryption. [1]</p>		None. Use Supplemental Procedures
4.3.7	Application object reuse	<p>Before shutdown, the application must delete/erase all temporary files, cache, data, and other objects it created during its execution. [2]</p>		None. Use Supplemental Procedures
4.3.8	Confidentiality of crypto. material	<p>The encryption facility invoked by the application must protect from disclosure all sensitive cryptographic material that is, keying material, private keys, and (if so indicated by the application's robustness) the cryptographic algorithm implementation. [2,18]</p>		None. Use Supplemental Procedures
4.3.9	Confidentiality of user identities	<p><i>If the identity of users must be protected from disclosure:</i></p> <p>The application must not [19]:</p> <ul style="list-style-type: none"> Reveal to external users or processes the identity of any user associated with any application session. Include within or append onto a data object an indicator of the identity of the data's creator or sender. 		Indirect mapping using all tools.

REQ #	QUICK REFERENCE	REQUIREMENT	VULN. ADR.	TOOL CATEGORY MAPPING
		<ul style="list-style-type: none"> Invoke any external process that includes within or appends onto a data object any indicator of the identity of the data's creator or sender. 		
INTEGRITY				
4.4.1	Integrity of transmitted data	<p><i>If the application transmits data over a high level of concern network:</i></p> <p>The application must use a NIST-approved or NSA-approved technology (as appropriate for the application's Mission Category) to implement a hash (e.g., Secure Hash Algorithm One [SHA-1]), checksum, or digital signature (e.g., DSS) of the data before transmission.</p>		None. Use Supplemental Procedures
4.4.2	Integrity of transmitted application code	<p><i>If the application is used to transmit application code (including some mobile code, see Section 4.6) over a high level of concern network with inadequately robust security protections:</i></p> <p>The application must invoke an approved digital signature technology to digitally sign the code prior to transmission. [1,18]</p> <p><i>NOTE: If desired, the application may also invoke an approved technology to apply a hash or checksum to the code before transmission.</i></p>		None. Use Supplemental Procedures
4.4.3	Integrity of stored data	<p><i>If the infrastructure on which the application runs does not adequately protect the integrity of data stored by the application:</i></p> <p>The application must invoke NIST- or NSA-approved technology (as appropriate for the application's Mission Category) to apply a hash, checksum, or digital signature to the data before storage.</p>		None. Use Supplemental Procedures
4.4.4	Integrity mechanism validation	<p><i>If the application is used to retrieve stored data or to receive transmitted data that have an integrity mechanism applied to them:</i></p> <p>The application must be able to validate the integrity mechanism, and must reject data for which the integrity mechanism validation fails.</p>		None. Use Supplemental Procedures
4.4.5	Validation of parameters	<p>The application must validate parameters before acting on them, and must reject all parameters that:</p> <ul style="list-style-type: none"> Are not formatted as expected by the application Do not fall within the bounds (length, numeric value, etc.) expected by the application 	V8	<ul style="list-style-type: none"> Web Application Developer General-Purpose
4.4.6	Notification of acceptable input	<p><i>If the application requires a user to input data:</i></p> <p>The application must inform the user of the expected characteristics of the input e.g., length, type (alphanumeric, numeric only, alpha-only, etc.), and numeric or alphabetic range.</p>		None. Use Supplemental Procedures
4.4.7	Validation of user input	<p>The application must validate all data input by users or external processes, and must reject all user input that:</p> <ul style="list-style-type: none"> Is not formatted as expected by the application Falls outside the bounds (e.g., length, range) expected by the application Contains HTML 	V7, V10, V11	<ul style="list-style-type: none"> Web Application General-Purpose

REQ #	QUICK REFERENCE	REQUIREMENT	VULN. ADR.	TOOL CATEGORY MAPPING
		<ul style="list-style-type: none"> Contains meta code Contains embedded Structured Query Language (SQL) queries that include illegal characters Contains any other type of unexpected content 		
4.4.8	Rejection of incorrect input	<p><i>If data input by users or external processes cannot be validated by the application:</i></p> <p>The application process that received the invalid input must (as appropriate given the purpose and robustness of the application):</p> <ul style="list-style-type: none"> Request the external user or process to reinsert the data, or Gracefully terminate the user process with an error message to the user indicating that the process is terminating as a result of an input error. 		Indirect mapping using Web Application, Developer, and General-Purpose tools.
4.4.9	Input validations by server	All user input validations must be performed by the server application, not by the client application (e.g., the application must not rely on browser JavaScript validation).		<ul style="list-style-type: none"> Web Application
4.4.10	Data containing active content	Application validation of user input data that contains active content (e.g., mobile code) must not result in the execution of the active content.	V9	<ul style="list-style-type: none"> Web Application
4.4.11	Application process integrity	<p><i>If the application updates data:</i></p> <p>The application's data update processes must operate correctly, and must not incorrectly reparse, inadvertently introduce errors to, or otherwise corrupt the data they update.</p>	V6	Indirect mapping using Web Application, Developer, and General-Purpose tools.
4.4.12	Integrity of transmitted application code	<p>If the application receives transmitted application code (e.g., mobile code):</p> <p>The application must find and validate the digital signature and any hash, checksum, or other additional integrity mechanism applied to that code before executing it. [1,18]</p> <p><i>If the code has no digital signature, or cannot validate the digital signature, hash, or checksum:</i></p> <p>The application must discard the code without executing it. (This discard must be audited.)</p>		None. Use Supplemental Procedures
4.4.13	Application configuration integrity	<p><i>If the underlying infrastructure does not adequately protect the integrity of the application's configuration and other parameter files from corruption or unauthorized modification by malicious processes or unauthorized users:</i></p> <p>The application must invoke a virus scanning tool to scan such files every time the application uses them. [2]</p> <p><i>NOTE: Administrators must keep virus signature files used by virus scanning tools up to date.</i></p>		None. Use Supplemental Procedures
4.4.14	Application executable integrity	<i>If a hash, checksum, or other integrity mechanism was applied to the application's executable code at installation time, before the application</i>		None. Use Supplemental

REQ #	QUICK REFERENCE	REQUIREMENT	VULN. ADR.	TOOL CATEGORY MAPPING
		<p><i>is executed:</i></p> <p>The process that validates that checksum or hash must be invoked before the application is executed to ensure that the application's current executable code state has not changed since the original integrity mechanism was applied.</p> <p><i>If this validation fails, the validation process must:</i></p> <ol style="list-style-type: none"> 1. Prevent the application from being launched 2. Notify the administrator that the application code needs to be replaced by an uncorrupted executable. 		Procedures
4.4.15	Time/date stamp of data modification	An application that modifies data must time/date stamp each data modification. [1]		None. Use Supplemental Procedures
4.4.16	Display of data time/date stamp	The application must display to each user who retrieves the data the time and date on which the data were last modified. [1]		None. Use Supplemental Procedures
4.4.17	Resolution of mode changes	Before it shuts down, the application must reverse any changes in the application's operating mode or state that occurred during its execution, and must return to its normal mode and state of operation.		None. Use Supplemental Procedures
4.4.18	Initialization of variables	<p><i>If the programming language in which the application is written does not automatically ensure that all variables are initialized to zero when declared:</i></p> <p>The application code must explicitly initialize all of its variables when they are declared.</p> <p><i>NOTE: "C" does not provide the necessary zero initialization of variables.</i></p>		<ul style="list-style-type: none"> • Web Application • Developer
4.4.19	Integrity of crypto data	The application must ensure that all sensitive crypto materials used by the application are protected from corruption or modification by unauthorized users or malicious processes. [19]		Indirect mapping using all tools.
4.4.20	Cryptokey revocation	The encryption facility invoked by the application must handle and respond correctly to Key Revocation Lists (KRL) issued by the cryptographic implementation and must not continue to use revoked keys. [19]		None. Use Supplemental Procedures
4.4.21	Certificate revocation	The PKI invoked by the application must handle and respond correctly Certificate Revocation Lists (CRL) issued by the PKI's certification authority and must not continue to honor revoked certificates. [19]		None. Use Supplemental Procedures
4.4.22	Signature of code	<p>If the application receives mobile code, interpreted (versus compiled) code, or other active content, it must not execute that code until it:</p> <ul style="list-style-type: none"> • Verifies that the code has been digitally signed • Validates the digital signature on the code. 		None. Use Supplemental Procedures
4.4.23	Use of hidden fields	<p><i>If an application Web page contains a hidden field:</i></p> <p>The application must validate the source of all HTML updates to the</p>	V21	• Web Application

REQ #	QUICK REFERENCE	REQUIREMENT	VULN. ADR.	TOOL CATEGORY MAPPING
		field and must reject any HTML field changes from unvalidated sources.		
AVAILABILITY				
4.5.1	Data availability	The application code must not contain errors, bugs, or vulnerabilities that could cause any executing process within the application to inadvertently delete or overwrite data, to incorrectly assign/change access permissions to that data, or to otherwise impinge on the data's availability.		<ul style="list-style-type: none"> • Web Application • Developer • General-Purpose
4.5.2	Server application availability	Server application code, regardless of Mission Category, should not include bugs, errors, or exploitable vulnerabilities that could cause the executing application to crash.		<ul style="list-style-type: none"> • Web Application • Developer • General-Purpose
4.5.3	Mission Category 1 client application availability	Mission Category 1 client applications* must not include bugs, errors, or exploitable vulnerabilities that could cause the executing application to crash.		<ul style="list-style-type: none"> • Web Application • Developer • General-Purpose
4.5.4	Maintenance of secure state	<p>If the application fails or is affected by an error, the crash/error must not cause the system to go into an insecure state. [1,2]</p> <p>Restart of the application must not cause the system to go into an insecure state.</p>		Indirect mapping using Web Application, Developer, and General-Purpose tools.
4.5.5	Application failure notification	<p>If an application process fails, the administrator must be immediately notified in one or more of the following ways (to be configured by the administrator) [2]:</p> <ul style="list-style-type: none"> • E-mail • Console message • Pager message. 		None. Use Supplemental Procedures
4.5.6	Secure application recovery	During recovery from a failure, the application must verify the integrity of its data, configuration files, parameters, etc., before executing. [2]		None. Use Supplemental Procedures
4.5.7	Application Denial Of Service (DOS)	Server application code, regardless of Mission Category, and Mission Category 1 client applications code must exclude bugs, errors, or exploitable vulnerabilities that could be exploited by a malicious user or program to launch a successful DoS attack against the application.		<ul style="list-style-type: none"> • Web Application • Developer • General-Purpose
4.5.8	Error handling and recovery	Application error handling and recovery capabilities must be robust enough that they cannot be overwhelmed by a flood of malformed arguments from malicious users or processes into a DOS state.	V15	<ul style="list-style-type: none"> • Web Application • Developer • General-Purpose
4.5.9	Missing files	<p>Before attempting to use any file or directory, the application must first verify that the file/directory exists on the system.</p> <p><i>If the file/directory is missing:</i></p>		<ul style="list-style-type: none"> • Web Application

REQ #	QUICK REFERENCE	REQUIREMENT	VULN. ADR.	TOOL CATEGORY MAPPING
		<p>The application must:</p> <ol style="list-style-type: none"> 1. Return an error message informing the user that the requested file/directory cannot be found 2. Gracefully terminate the user process through which the user requested the missing file/directory, and the server process that searched for that file/directory. 		
4.5.10	Key recovery	The encryption facility invoked by the application must perform the key recovery processes required by the cryptographic implementation.		None. Use Supplemental Procedures
ACCOUNTABILITY				
4.6.1	Audit/event logging mechanism	The application must log all security-relevant events (configured by the administrator) to its own secure audit file, or transmit its log data to an external audit facility. [1,2,18]		<ul style="list-style-type: none"> • Database <p>Indirect mapping using General-Purpose and Web Application tools.</p>
4.6.2	Configurable audit/log parameters	The audit facility used by the application must allow the administrator to select the events to be logged and the information to be captured about each event. [1,2,18]		<ul style="list-style-type: none"> • Database
4.6.3	Events to be audited/logged	<p>The application must log the following types of events to its audit facility, at a minimum [1,2,18]:</p> <ul style="list-style-type: none"> • Startup and shutdown • Authentication • Authorization/permission granting • Actions by trusted users • Process invocation • Data access attempt • Data update • Data deletion • Input validation • Establishment of network connection • Data transfer • Application configuration change • Application of confidentiality or integrity labels to data • Override or modification of data labels or markings • Output to removable media • Output to a printer. 		<ul style="list-style-type: none"> • Database <p>Indirect mapping using General-Purpose and Web Application tools.</p>

REQ #	QUICK REFERENCE	REQUIREMENT	VULN. ADR.	TOOL CATEGORY MAPPING
4.6.4	Binding of UserID to audit record	The audit facility used by the application must bind the individual ID of the user causing (or associated with) the audited event to the audit record for that event. [2]		<ul style="list-style-type: none"> Database Indirect mapping using General-Purpose and Web Application tools.
4.6.5	Audit information captured by classified applications	<p><i>If the application handles classified data:</i></p> <p>Each audit record must include the following information (as relevant for the type of event) [1,2,18]:</p> <ul style="list-style-type: none"> UserID of user or process ID of process causing the event Successful or failure of attempt to access a security file Date and time of the event Type of event Success or failure of event Seriousness of event (violation)* Successful or failure of login attempt Denial of access resulting from excessive number of login attempts Blocking or blacklisting a UserID, terminal, or access port, and the reason for the action Data required to audit the possible use of covert channel mechanisms Privileged activities and other system level access Starting and ending time for access to the application Activities that might modify, bypass, or negate safeguards controlled by the system Security-relevant actions associated with periods processing, or the changing of security labels or categories of information 		Indirect mapping using Database, Web Application, and General-Purpose tools.
4.6.6	Audit information captured by sensitive and nonpublic access applications	<p><i>If the application handles sensitive or unclassified but not-publicly releasable data but no classified data:</i></p> <p>Each audit record must include the following information (as relevant for the type of event) [1,2,18]:</p> <ul style="list-style-type: none"> UserID of user or process ID of process causing the event Success or failure of attempt to access security file Date/time of event Type of event Success or failure of event 		Indirect mapping using Database, Web Application, and General-Purpose tools.

REQ #	QUICK REFERENCE	REQUIREMENT	VULN. ADR.	TOOL CATEGORY MAPPING
		<ul style="list-style-type: none"> Seriousness of event (violation) Success or failure of login attempt Denial of access resulting from excessive number of login attempts Blocking or blacklisting of UserID, terminal, or access port, and reason for the action Activities that might modify, bypass, or negate security safeguards controlled by the application 		
4.6.7	Audit information captured by public access applications	<p><i>If the application handles only publicly releasable data:</i></p> <p>Each audit record must include the following information (as relevant for the type of event) [1,2,18]:</p> <ul style="list-style-type: none"> UserID of user or process ID of process causing the event Success or failure of attempt to access security file Date/time of event Type of event Success or failure of event Seriousness of event (violation). 		Indirect mapping using Database, Web Application, and General-Purpose tools.
4.6.8	Protection of audit records	The audit facility used by the application shall ensure that the application's audit records are protected from deletion or unauthorized disclosure/modification. [1,2]		Indirect mapping using Database, Web Application, and General-Purpose tools.
4.6.9	Audit trail "fill thresholds"	<p>The audit facility used by the application shall enable the administrator to set the audit trail "fill thresholds" as follows [2]:</p> <ol style="list-style-type: none"> A threshold that indicates the audit trail is some percentage full, which shall trigger a notification to the administrator that the file should be archived and purged. A threshold that indicates the audit/log file is full, which shall trigger one of the following events (configurable by the administrator): <ul style="list-style-type: none"> Graceful shutdown of the application, or Suspension of user processing, or Overwriting of the oldest audit records, or Termination of auditing, or Increase of storage space allotted for audit records (to an amount configurable by the administrator) 		None. Use Supplemental Procedures
4.6.10	Audit failure	<p>If the audit facility used by the application fails, one of the following events (configurable by the administrator) must occur [2]:</p> <ul style="list-style-type: none"> Shutdown of the application, or 		None. Use Supplemental Procedures

REQ #	QUICK REFERENCE	REQUIREMENT	VULN. ADR.	TOOL CATEGORY MAPPING
		<ul style="list-style-type: none"> Suspension of user processing, or Notification of the administrator, or Automatic restart of the audit facility 		
4.6.11	Security violation notifications	<p>The audit facility used by the application must [1]:</p> <ol style="list-style-type: none"> Immediately alert the security administrator of all security violations and unusual or suspicious activity that might indicate a security violation. Gracefully shut down the application if the event is considered serious* enough to warrant it. 		Indirect mapping using Database, Web Application, and General-Purpose tools.
4.6.12	Audit trail viewing and reporting tool	The audit facility used by the application shall include a tool that enables the administrator to view the application's audit records, and to report against them. [1]		None. Use Supplemental Procedures
NON-REPUDIATION				
4.7.1	Digital signature of created and transmitted data	<p><i>If the application requires non repudiation by the creator or sender of data or cryptokeys:</i></p> <p>The application must invoke a NIST- or NSA-approved digital signature technology (e.g., DSS) appropriate to the application's Mission Category to enable the creator/sender to digitally sign the data/keys the application is used to create or transmit over the network. [1,2]</p>		None. Use Supplemental Procedures
4.7.2	Digital signature of received data (proof of delivery)	<p><i>If the application requires non repudiation by the recipient of data sent over the network:</i></p> <p>The application must invoke a NIST- or NSA-approved digital signature technology (e.g., DSS) appropriate to the application's Mission Category to enable the recipient to sign received data. [1]</p>		None. Use Supplemental Procedures
4.7.3	Digital signature validations	The application must invoke a digital signature validation facility to validate all digital signatures applied to data or cryptokeys it receives over the network or retrieves from a database or directory. [1]		None. Use Supplemental Procedures
4.7.4	Protection of digital signature security data	The application must protect from tampering and inappropriate disclosure the cryptokeys and certificates it uses for digital signature processing. [1]		Indirect mapping using all tools.
4.7.5	PK-enabling of e-mail applications	All DoD e-mail applications (including Web e-mail applications) must include or invoke a digital signature facility that uses Class 3 certificates; this facility must accommodate use of Class 4 certificates by the deadline specified in DoD PKI Policy, with minimal modification to application code. [6,7]		None. Use Supplemental Procedures
MOBILE CODE				
4.8.1	Category 1 and 2 mobile code source	<p><i>If the Web server application (or other server application) acts as the source for Category 1 or Category 2 mobile code:</i></p> <p>The application must digitally sign the mobile code before releasing it using a PKI code signing certificate approved by NIST or NSA (depending on the Mission Category of the application) to sign the</p>		None. Use Supplemental Procedures

REQ #	QUICK REFERENCE	REQUIREMENT	VULN. ADR.	TOOL CATEGORY MAPPING
		code. [1,8] <i>Exceptions to this requirement:</i> <ul style="list-style-type: none"> <i>If the responsible CIO has signed a written waiver allowing the use of commercial certificates for mobile code signing, the application may release Category 1 or Category 2 mobile code signed using commercial certificates.</i> <i>If the responsible CIO has signed a written waiver allowing the use of Category 2 mobile code without code that has not been signed, the application may release Category 2 mobile code without a digital signature.</i> 		
4.8.2	Category 1 and 2 mobile code execution	Before executing Category 1 or Category 2 mobile code, the browser (or other client application) must validate the digital signature on the mobile code to ensure that the code originated from a trusted source. [1,8]		None. Use Supplemental Procedures
4.8.3	Category 2 mobile code execution	Category 2 mobile code must be implemented in a way that ensures that it executes in a constrained environment without access to local OS and network resources (e.g., file system, Windows registry, and network connections other than to its originating host). [1,8]		None. Use Supplemental Procedures
4.8.4	Category 2 mobile code notification	The browser (or other client application) must be configurable to warn users when the application is about to execute Category 2 mobile code. [1,8]		None. Use Supplemental Procedures
4.8.5	Category 3 mobile code	The application may act as a source for, or may execute, Category 3 mobile code after a risk assessment has been performed, and appropriate risk mitigation safeguards and countermeasures have been implemented. [1,8]		None. Use Supplemental Procedures
4.8.6	Emerging mobile code technology	The application must not act as a source for, or execute, emerging mobile code technology unless a written waiver has been granted according to DoD Mobile Code Policy Section 1.4.3. [1,8]		None. Use Supplemental Procedures
4.8.7	Mobile code in e-mail messages	An e-mail client application must disable/prevent (or interface in trustworthy way with another program that can disable) the execution of mobile code in message bodies or attachments. [1,8]		None. Use Supplemental Procedures
4.8.8	E-mail client mobile code notification	An e-mail client application must be configurable to issue a warning to the user, before opening an e-mail attachment, that the attachment about to be opened may contain mobile code. [1,8]		None. Use Supplemental Procedures